ABSTRACT

NONMONOTONIC CRYPTOGRAPHIC PROTOCOLS

by

Aviel David Rubin

Co-Chairs: Adjunct Associate Professor Peter Honeyman & Professor Bernard Galler

A new Kerberos service, *khat*, is provided to add the functionality of long-running jobs to an authenticated environment. The protocol used by *khat* is nonmonotonic. That is, the knowledge of principals in the protocol may increase or decrease. A survey of existing protocol analysis techniques reveals that no method exists for analyzing nonmonotonic cryptographic protocols.

A method for specifying and analyzing nonmonotonic cryptographic protocols is provided. An advantage of the specification technique is that protocols are specified at a level close to the actual implementation. The analysis technique uncovers several known flaws in protocols that are used as benchmarks to test many analysis systems. In addition, it discovered a flaw in the original, published version of the *khat* protocol, that led to a revised version.

The analysis technique offers several advantages over existing ones, even for monotonic protocols. Protocols can be analyzed at any stage in their development. Previous techniques generally apply to entire protocols. Also, a protocol can be tested against known attacks. Another advantage is that the protocol designer is required to state assumptions, such as trust in a server, in the specification. These assumptions are often implicit in current methods, often leading to faulty implementations.

In practice, replay attacks are avoided with nonces or timestamps. Nonces are large, random numbers that are used to guarantee that a message was generated during the current run of a protocol. Unfortunately, improper use of nonces can lead to flaws. The new technique prevents many of these problems by restricting the use of nonces to linking one response to a challenge. Improper use of nonces is detected.

Another type of flaw arises when an intruder can masquerade as another principal by imposing a public key upon an unsuspecting participant. It is shown that binding keys to principals helps avoid this problem. The new technique requires that all keys be bound to principals, and improper bindings are detected.

The observation is made that restricting the freeness of variables in a protocol, such as nonces and keys, leads to increased security.

NONMONOTONIC CRYPTOGRAPHIC PROTOCOLS

by

Aviel David Rubin

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in The University of Michigan
1994

Doctoral Committee:

Adjunct Associate Professor Peter Honeyman, Co-Chair
Professor Bernard A. Galler, Co-Chair
Assistant Professor Sean Coffey
Associate Professor Larry Flanigan
Assistant Professor Atul Prakash

"Here's ado
To lock up honesty and honour from
The access of ... visitors."

William Shakespeare

Dedicated
to my future wife, Ann
and to my parents,
Abba and Carol Rubin

ACKNOWLEDGEMENTS

I would like to thank Peter Honeyman for his guidance and advice, and most of all, for teaching me how to be a researcher. I am deeply indebted to Peter for providing me with an ideal research environment, and for taking so much interest in my work.

I thank Bernie Galler for recommending computer science to me, and for showing me how interesting it can be. I would like to especially thank Bernie for adding a special quality to my nine years at Michigan, and always showing me a new perspective on life. Bernie was always there for me through some difficult transitions in my life, and his intelligence and father-like support helped me get through them.

I thank Atul Prakash for suggesting that I explore protocol analysis. This suggestion is directly responsible for shaping the direction of my research. I also thank the other members of my committee for agreeing to serve, reading my papers, and providing me with useful ideas. In addition, I thank Kevin Compton for all the help with theory throughout my graduate career.

I wish to thank my fiancee Ann for helping me enjoy the good moments of life and helping me overcome the difficult ones. I thank Mom, Dad, Grandma, Bube, Rachel, Ron, Tova, Uri, and Yaacov, for knowing all along that I could do it. Also, I thank my parents for their constant advice and motivation.

I would also like to thank Mary Jane Northrop and Edna Brenner for help with editing as I finished various parts of this thesis. I thank Dan Muntz, Larry Huston, David Snearline, and Scott Dawson for helpful suggestions during the implementation phase of *khat*. Finally, I thank Terri, Kati, Elaine, Becky, CJ, Stolar, Bill, Ted, Wanda, Masud, Seshu, Chris, Janani, Lee, Norman and everyone else at the Center for Information Technology Integration for their support and friendship.

This work was partially funded by IBM.

TABLE OF CONTENTS

LIST OF APPENDICES

Appendix

LIST OF FIGURES

Figure

CHAPTER 1

INTRODUCTION

The field of computer science consists of many interesting subfields. While research topics such as parallel processing, distributed networking, and other popular areas receive much attention, an important, yet often neglected subject is computer security.

The advent of distributed networks revolutionized computer science, and the communication between computers became as important as the processing. In a modern network, some computers serve as client workstations, while others are designated to provide services. Whenever possible, these services are transparent to the user, who is unaware of the communication that takes place between his machine and the remote server. For example, files are usually maintained on a file server, and clients request them as needed. The user need not know if the files are on the local disk.

Before distributed networks, files resided on the local machine, and a malicious intruder had to compromise the computer to access them. However, in today's systems, files travel on a network composed of wires and phone lines that are easily accessible to the outside world. An intruder can read them, modify them, and even destroy them. The security of information is a difficult problem that gains in importance every day, as computer networks become ubiquitous. Computer security is becoming especially important in light of the projected information superhighway.

The science of cryptography is the foundation of computer security. An encryption algorithm applies a combination of substitutions and transpositions to the bits of a data block and a key. There are two classes of algorithms. In *symmetric* systems, the key is secret, and it is only known to the parties who encrypt or decrypt [58]. Also, there are two separate algorithms for encryption and decryption

respectively. In *asymmetric*, or *public* key systems, there is only one algorithm [58]. In addition, there are two keys that are inverses of each other. Encrypting with one requires the other for decrypting, and vice versa. Usually, one key is called the *private* key, and it is only known to one user. The other is called the *public* key, and it is universally available.

The security of a system lies in the length of the keys, and it is widely accepted that security by obscurity is obsolete [58]. Cryptography is used to hide information (secrecy), to guarantee that information has not changed (integrity), and to guarantee the origin of information (authentication).

The use of cryptography for secrecy, integrity, and authentication in a distributed system requires an environment in which each principal is in possession of a secret key. This is called an authenticated environment. Depending on the type of cryptography in use, each principal may also have a public key. The distribution of these keys is a non-trivial problem (see Chapter 5). Principals in an authenticated environment use cryptographic protocols to communicate and share information. These protocols are designed to preserve the three security properties mentioned above.

One example of cryptography in an authentication protocol is in a *challenge* and *response*. When a principal, A wishes to verify that he is communicating with another principal, B, with whom he shares a secret key, K, he sends a message to B. This message servers as a challenge. B encrypts the message using K, and sends the new message to A. If A believes that the only other principal knowing K is B, then the correct encryption of the message indicates that it came from B.

Kerberos [64], developed at MIT, is an authentication system that is in widespread use. In this system, each principal shares a secret encryption key with the Kerberos server, a machine that must be kept physically secure. The secret key for a user is created by applying a one-way hash function to his password. This function has the property that it cannot be easily inverted. The existence of a secret key in two separate locations can be viewed as a secure channel in which encrypted messages can be sent.

Using Kerberos, secure channels can be established between any two principals. When two principals wish to establish a secure connection, the Kerberos server creates a *session key*, and distributes it to them using existing secure channels. A session key is a key that is only used for one communication session and is then dis-

carded. This process requires cryptographic protocols that are known to all parties involved.

In Kerberos, session keys are distributed in *tickets*. When principal A wishes to establish a secure channel with principal B, he requests a ticket for B from the Kerberos server. This ticket contains the session key, K, along with an instruction that it be used with A. The Kerberos server sends a copy of K to A, along with a ticket that is forwarded to B. The ticket represents an unforgeable proof of identity for A. The ticket contains a lifetime to limit its exposure. Therefore, Kerberos relies on loosely synchronized clocks.

Kerberos issues tickets for services. If a principal does not possess a valid ticket, he has no access to any resources that require authentication. As most important service providers, such as the file server, require authentication, a principal without valid tickets cannot accomplish much.

Unfortunately, the limited lifetime of tickets creates a problem. Because tickets expire, users must manually renew them. Also, there is no way to schedule a batch job for some time in the future because without tickets, the job will have no access to needed resources. We provide a solution to the problem of long-running jobs in an authenticated environment. The *khat* system presented in Chapter 2 is based on the Unix *at* command. A user schedules a job with *khat*. The job is then stored on a secure machine until it is time for it to run. When the right time arrives, the job and a ticket are sent to the client's machine and begins execution. The ticket is for a special service called the ticket granting service that issues tickets for services to authenticated clients.

A protocol is said to have a *security flaw* if a malicious intruder can manipulate the messages to his advantage, can cause the principals involved to reach incorrect conclusions, or can prevent the principals involved from reaching desired conclusions. An example of this is the replay attack. If a message cannot be distinguished from a previous message, then it can be used to fool a principal into accepting it again. Such an attack does not require any knowledge of the encryption being used.

Replay attacks can be avoided by use of *timestamps* or *nonces*. A nonce is a large random number that is used only once. If a principal creates a nonce, N, at time t, then any message that contains N must have been created after t. The most common use of nonces is to recognize a message as a response to a challenge. The

challenge contains a nonce, which may be sent in the clear. An encrypted message that contains the same large number ensures a fresh response. That is, the response is not a replay of a previous message. Any information that contains a guarantee of timeliness is considered fresh, and any information that is not fresh is subject to a replay attack.

There are many other types of flaws in protocols [1]. New ones are constantly being discovered, and it is probably impossible to name all of the possible types of flaws in cryptographic protocols. Some flaws result from the interaction between the messages and the underlying cryptosystem, whereas others result from the messages themselves [45]. Once a type of flaw is discovered, it is relatively easy to use formal methods to check a protocol for this flaw. However, in practice, many protocols exist that contain known types of flaws.

Experience has shown that it is not always necessary to break the encryption algorithms to compromise a system. It may be possible for an active intruder to manipulate cryptographic protocols to his advantage. In fact, some protocols that are in use today contain subtle flaws that took years to discover. Therefore, formal methods are needed to analyze the security of protocols from malicious attacks. Chapter 3 discusses previous approaches to formal methods for cryptographic protocol analysis. The most successful approach uses logics specifically developed for the analysis of knowledge and belief.

The protocol used by *khat* cannot be analyzed using existing formal methods. *Khat* relies on the user's workstation erasing the batch job. The only copy resides on the secure server. There is no way to specify this using existing methods. We define the class of protocols where the participants' knowledge is strictly increasing as *monotonic*. If the knowledge of the principals in a protocol can increase or decrease, we say that the protocol is *nonmonotonic*. It is clear that the *khat* protocol is not monotonic.

There are other protocols that are not monotonic as well. For example, Thurasingham [72] describes a database management system that handles data that can be deleted. Protocols for interacting with this database are nonmonotonic because there are principals whose knowledge increases and decreases. There are many such protocols in use.

Protocols that are nonmonotonic may require that a principal delete some

information. Thus, the knowledge of principals is not monotonic. The need to formally analyze protocols such as *khat*, and others like it, prompted the development of a new technique for specifying and analyzing cryptographic protocols. This is presented in Chapters 4 and 5. The technique was used to discover a flaw in the original published version of the *khat* protocol [54]. This further demonstrates the need for formal tools to evaluate cryptographic protocols.

In the specification technique, a protocol designer chooses from a set of actions to define the steps in the protocol. Actions that are merely assumed in other specification methods, such as encryption and decryption, are stated explicitly. An advantage to this method is that the specification can be implemented directly. By reducing the level of abstraction between the specification and the implementation, the risk of unexpected flaws is lowered.

There are other advantages to the new method even for monotonic protocols. The protocol designer can interact with the analysis to check for known flaws. Also, the knowledge of an intruder can be modeled to see what information can be learned from passive or active interference. Another feature concerns the assumptions of the protocol, which must be stated explicitly in the specification. This avoids confusion when the protocol is implemented.

The actions that specify a protocol are used to represent the knowledge of the principals. The analysis technique uses inference rules to reason about beliefs. Another feature is designed to prohibit improper use of nonces. A special construct, called a LINK, links challenges and responses that use nonces, to require that nonces are only used once. If a nonce is used again, then the message containing it is ignored. In addition, the analysis technique detects the use of a nonce for any purpose other than linking a challenge to a response. This is discussed in Chapter 4.

The technique also helps detect flaws of another type. Abadi and Needham [1] show how one principal can masquerade as another by imposing a public key on an unsuspecting participant. The flaw is removed by attaching a principal's name to every key. We call this *binding* the keys. Each principal is associated with a *bindings* set, that enforces the requirements of binding keys to principals. Chapter 5 covers this topic in detail.

The binding of keys to principals, and the linking of nonces to unique responses leads to an observation about the variables, such as keys and nonces, in

a system. In both cases, restricting the variables reduces the possibilities of flaws. Thus, we conclude that eliminating the freeness of variables in cryptographic protocols increases security.

CHAPTER 2

LONG-RUNNING JOBS IN AN AUTHENTICATED ENVIRONMENT

2.1 Overview

In strong authentication systems, users may obtain access to secure system resources only when in possession of valid credentials. These are issued with limited lifetimes; their renewal requires a user to enter his password. We have developed a system called *khat* with which a user may schedule a batch job to be run at a later date in the current environment. The batch job is stored on a secure machine, and sent and received in encrypted form. When it is time for the job to run, the server generates credentials for the originating user and sends them encrypted to the machine on which the job will run. The user is given an option to specify that tickets should be continually generated for the job until its execution has completed.

2.2 Introduction

Adaptations of Needham and Schroeder's authentication system [48] are a boon for establishing secure services in distributed systems. One such adaptation is the Kerberos Authentication system [64] of MIT's Project Athena. An unfortunate byproduct of building Kerberos-based systems is a loss of functionality, such as long-running jobs. In this chapter, we address this weakness and offer a solution.

Before Kerberos, UNIX authentication was coterminous with a login ses-

sion. In the Kerberos system, tickets[1] expire, so that a compromised[2] ticket does not allow an impostor to masquerade as an authenticated user forever. Consequently, users are forced to reauthenticate on a regular basis, usually about once a day, to acquire fresh tickets.

For a Kerberos user to submit and execute a long-running batch job that employs secure system resources, he must either physically reauthenticate whenever tickets are about to expire, or enter a password into a script. Similarly, it is impossible to schedule a batch job to run at a distant future date and be authenticated as the user.

This problem has been recognized as a difficult one. Lampson *et al.* state that "it is a tricky exercise in balancing the demands of convenience, availability, and security" [35]. They further state that "the basic idea is to have a single highly available agent for the user that replaces the login workstation and refreshes credentials for long-running jobs." This approach is the one we take in solving the problem for Kerberos-based systems.

This chapter presents the original version of *khat*. The analysis technique presented in Chapter 4 uncovered a subtle security flaw in the protocol. The corrected protocol, which is the current one being distributed, is found in Appendix D.

2.3 KHAT: A New Kerberos Service

This section describes a new service: *khat*, based on the UNIX *at* command.[3] In a later section, we critique the design of *khat* and offer suggestions for improvement.

Like *at*, which is used to schedule batch jobs at a specific time and date, *khat* offers a batch service. The principal difference between the two is that *khat* provides continuous Kerberos authentication to the batch job while it runs.

---

[1] Kerberos credentials.

[2] *E.g.,* stolen.

[3] The name is taken from Kerberos and *at*. The additional letter stems from a tradition that we don't completely understand, but slavishly follow.

2.3.1 Overview of KHAT

When a user wishes to schedule a batch job, he issues the *khat* command with a syntax very similar to the UNIX *at* command. A spool file for the batch job is created, containing, among other things, the user's name, his current working directory, and his shell environment. So far, this is identical to *at*.

Next, *khat* requests a ticket for the *khat* service from the Kerberos ticket granting server on behalf of the user. This step is shown in Figure 2.1. The *khat* ticket

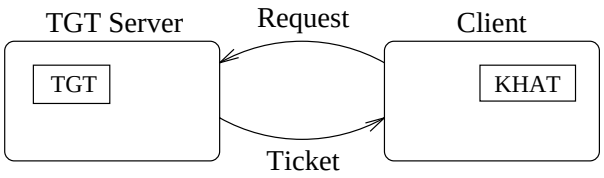

Figure 2.1: The client sends the ticket granting ticket (TGT) server a request for a *khat* ticket. The response includes a ticket that enables the *khat* server to authenticate the client.

is then sent to a *khatd* server. The server must run on a secure machine. The client and server then mutually authenticate [64], as shown in Figure 2.2. After mutual

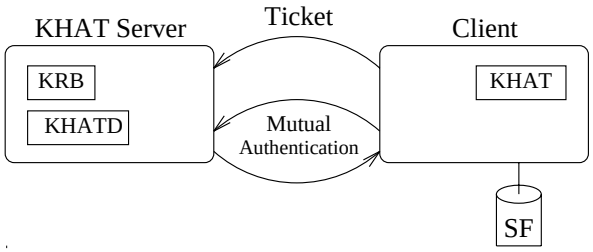

Figure 2.2: The client has generated a spool file (SF) for the *khat* job, which it stores on the local disk. It sends a *khat* ticket to the server (KHATD), and the client and server mutually authenticate. Note that the Kerberos server (KRB) is running on the same machine as *khatd*.

authentication, the client and server share a DES [47] key, which we denote K, as shown in Figure 2.3. The *khat* client uses K to seal the spool file. This hides the details of the batch request from prying eyes, as well as assuring its integrity. The encrypted spool file is then sent to the *khatd* server, as shown in Figure 2.4. Along with the spool file, the client sends information such as the time and date for the job to run, the user's environment, *etc.* The *khatd* server receives the spool file from the client, unseals it, and stores the file away until it is time for the job to run. At this

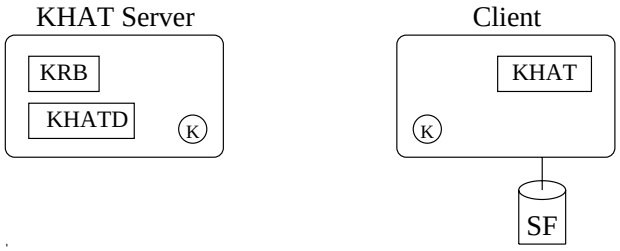

Figure 2.3: Initially, the spool file resides on the client machine. The client and server share a secret key, K.

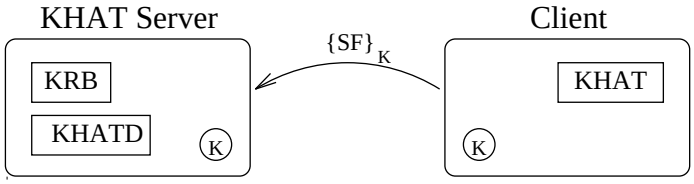

Figure 2.4: The client sends the spool file to the server, sealed under the session key, K.

point, the client discards its copy of the spool file and listens to a well known port for activation. For reasons discussed later, the *khatd* server runs on a Kerberos master or slave machine. This blocked configuration is depicted in Figure 2.5. Periodically,[4] the

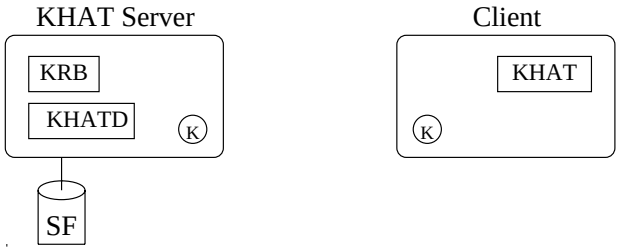

Figure 2.5: The spool file is stored on the secure server machine, and is purged from the client machine.

server checks to see if any job is scheduled to be run. When the time for the job to run arrives, the server uses the Kerberos database to construct a ticket for the user. It then seals this ticket along with the spool file, and sends them back to the client machine, as shown in Figure 2.6. The client can use the TGT to obtain tickets for other services. If the job will run for longer than the life of the ticket, or if the user suspects this may be the case, *khat* offers an option to renew tickets, in which case the server sends new tickets to the client as long as the job is running. Of course, care is

[4]Once a minute, in our implementation.

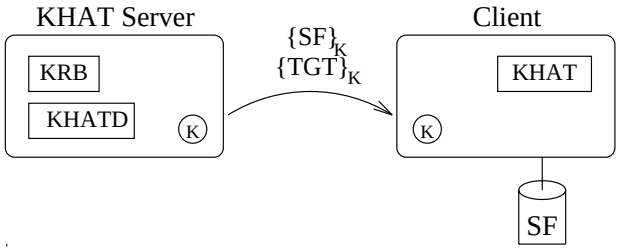

Figure 2.6: When it is time for the job to run, the encrypted spool file and a ticket granting ticket (TGT) are sent to the client under the session key.

taken to ensure that the job is still running. (This proves to be a difficult problem.)

After the job terminates, the spool file is removed from the client machine, and the *khat* process exits. A message is then sent to the server machine so that the *khatd* process can exit too. Details follow in Section 2.4.6.

2.3.2 Implementation of the KHATD server

The server program, *khatd*, runs as root. Moreover, the server machine contains a master or slave copy of the Kerberos database; any program running on such a machine must be trustworthy. The *khat* program runs on the client's host machine. It also runs as root, but care is taken to make sure that the user's job is not able to obtain a higher privilege than it should. The program uses the UNIX *setuid* facility to ensure that the user's batch job runs as that user. However, the actual *khat* program runs as root because it must perform operations that require special privileges, such as setting group IDs, deleting files, and copying files to and from the local disk.

When the *khatd* server needs to issue tickets for a user, it sends a request to the Kerberos server. The Kerberos server returns a user ticket encrypted under that user's secret key. Unfortunately, the client machine does not necessarily have a user authenticated to it when this happens, so the user's secret key may not be available. Thus, the user's ticket issued by the Kerberos server is not readable.

To decipher the user ticket, the *khatd* server uses the Kerberos database to access the user's secret key. User keys are stored encrypted under a master key in the Kerberos database, so *khatd* must first use the Kerberos master key. Once it has decrypted the user's secret key, *khatd* decrypts the user ticket received from the Kerberos server. Finally, *khatd* changes the client address in the ticket to that of the

target host that will run the job. The ticket is then ready to be sent to the target host. The encryption of the ticket before it is sent over the network is the topic of Section 2.4.5.

When a user issues the *khat* command, the TGT is used to request a *khat* ticket. After mutual authentication, the client receives a ticket and can then communicate with the server. Thus, *khat* behaves as any other Kerberos service, with one exception: to construct the ticket, *khatd* has to access the Kerberos database and the Kerberos master key. Thus, *khatd* must run on a Kerberos master or slave machine.

2.4 The Theory Behind KHAT

It seems contradictory to provide authentication for an absent principal. By definition, authentication means that a principal proves his identity, seemingly an impossible task if he is no longer present. Thus, to schedule a long-running job, or one to be run at a later date, a principal must leave something around so that possession of the thing is equivalent to authentication for that principal. A similar idea, called delegation, is discussed by Lampson *et al.* [35]. The authors define the "speaks for" relationship and provide rigorous definitions and proofs based on a set of axioms they define in the paper.

Ideally, we would like the user's batch job to delegate authority to the workstation, saying that the workstation speaks for the user. In general, though, we are dealing with the domain of untrusted workstations. Many workstations reside in public sites where many different users have access to them at all times. It is a fundamental assumption that nothing on such a workstation can be trusted. However, some compromises must be made to provide for authenticated long-running jobs; we elaborate on this theme in the next section.

2.4.1 The authentication problem for a vacant workstation

We call a workstation *vacant* whenever a given user's task must be run there while that user is not logged in. In the previous section, we assumed that nothing on the workstation could be trusted. The reason for this is straightforward: we must allow for the possibility that a user might obtain root privileges, *e.g.*, by booting the

machine into single-user mode, whereupon the privileged user might replace any or all utilities on the workstation, including the operating system image itself. The only objects on a public workstation safe from such attack are those that are encrypted. Yet, we take it as given that the encryption key may not reside on the workstation, even if well hidden.

Experts dismiss systems that hide cryptographic algorithms or protocols (a.k.a. "security through obscurity"). Kahn [30] cites Kerckhoffs' classic treatise on military security [34]. Saltzer and Schroeder [55] reflect a more modern view in describing "open design" as one of the basic principles of information protection:

> The design should not be secret. The mechanisms should not depend on the ignorance of potential attackers, but rather on the possession of specific, more easily protected, keys or passwords. This decoupling of protection mechanisms from protection keys permits the mechanisms to be examined by many reviewers without concern that the review may itself compromise the safeguards. In addition, any skeptical user may be allowed to convince himself that the system he is about to use is adequate for his purpose. Finally, it is simply not realistic to attempt to maintain secrecy for any system which receives wide distribution.

Voydock and Kent amplify this perspective: "data encryption is the fundamental technique on which all communications security measures are based" [78]. Cloaking information does not protect it.

Therefore, in a secure, distributed authentication system, data must travel across the network encrypted; for two peers to communicate this way, they must share a secret. Thus, the user must place something on the workstation which the server can later use for mutual authentication.

Lampson *et al.* describe a mechanism whereby a vacant workstation could share a secret [35]. Their method requires that a machine possess a private key stored in nonvolatile memory. In addition to the private key, certificates and other rules must be stored on the boot ROM.

Aside from the fact that our workstations do not contain this information in ROM, Lampson's method requires a public key system, which is not compatible with Kerberos. At some future date, it may be possible to authenticate a workstation, whereupon it will not be necessary for the client to leave anything on the machine.

2.4.2 How KHAT authenticates to a vacant workstation

For the server to send an encrypted spool file and tickets back to the client's host machine, some shared secret must be left on the workstation. The creation and responsibility of this secret is illustrated in Figure 2.7. To leave this secret on the workstation, a new random key is generated, which we denote N.[5] K, the *khat* session key, is encrypted with N and stored on the client machine. N is then sent encrypted under K to the server. Note the symmetry here: the client holds $\{K\}_N$ but sends the server $\{N\}_K$.

After sending the spool file and other information to the server, the client erases all of this information from disk and from memory. All that the workstation keeps is $\{K\}_N$, the session key from the original *khat* ticket encrypted under N, accessible only to root on the client machine. When it is time for the job to run, the server sends N to the workstation in the clear, followed by the spool file and tickets, sealed under K. The workstation then unseals the original session key and uses it to decrypt the spool file and tickets.

By itself, eavesdropping on the network does not expose the user: the only function served by N is to unseal the key on the workstation. Once the session key is unsealed, the spool file and tickets sent across the network can be decrypted by the *khat* agent sitting on the workstation.

2.4.3 Risks of running KHAT with a vacant workstation

In this section, we discuss the risks involved in running *khat* on a vacant workstation. If the workstation is rebooted, then the process memory is lost. Although a denial of service results, this can be reported back to the user and no real harm is done beyond termination of the job. If an impostor manages to gain control of the machine without erasing the memory, and examines memory to find the secret key, this will give no advantage, as the secret key is encrypted with N. In fact, a few other safeguards are in place. The impostor has no idea when a job is scheduled to run, as all such information has been kept secret and no longer resides on the workstation.

---

[5] For *nonce*.

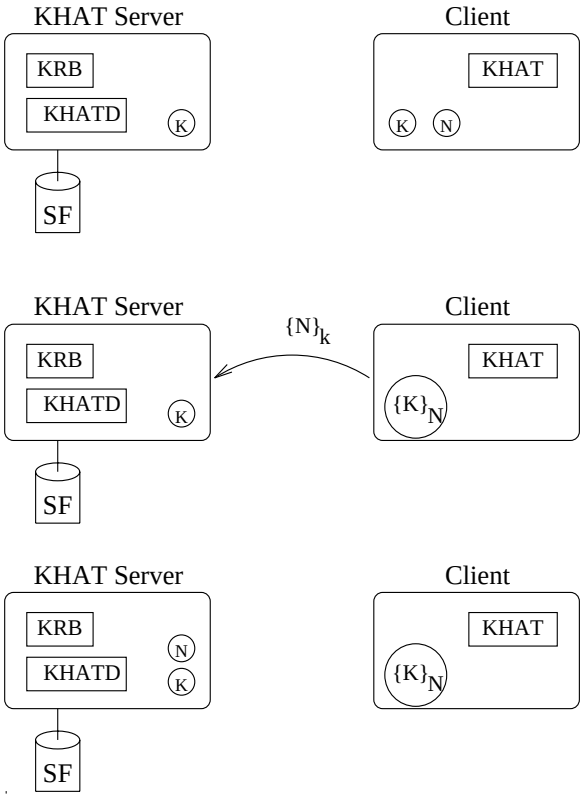

Figure 2.7: The first step is illustrated in the top diagram. The client generates a random key, N. Then, as shown in the next diagram, N is used to encrypt the session key, K, which remains on the client machine. Then, N is sent to the server under the session key. Finally, as shown in the bottom diagram, K and N are stored on the server. When it is time for the job to run, N is sent to the client to unseal $\{K\}_N$.

To compromise a job, an impostor must acquire the session key. But K is never exposed over the network, so the impostor must compromise the workstation itself. Even here, this must be accomplished either while the *khat* service is being requested, or while the user's job is being run. Even if $\{K\}_N$ is obtained from the workstation's process memory and N obtained through network eavesdropping, the impostor will be left with a ticket usable only on the target host. In summary, an impostor must completely compromise the workstation to affect the batch job. We know of no system that can run jobs securely on such a machine.

To deny service, an impostor can simply reboot a workstation, but *khat* can notify the user of unsuccessful batch jobs. As long as users understand this risk, they can choose whether to use the *khat* service. *Khat* does not compromise the security of people who do not use it.

2.4.4 The single user approach

The vacant workstation problem was addressed by Treese at MIT, where access to an Athena workstation is limited to one user at a time [74]. Treese reports that "experience has shown that this is an acceptable limitation." Placed in our context, workstations could prohibit any login while any *khat* job is scheduled. This would make *khat* client machines much more secure. However, this would allow a denial of service attack, wherein a malicious user schedules a *khat* job that monopolizes a workstation. Conversely, a user could log in and prevent a scheduled *khat* job from running. Both these forms of denial of service can be detected. In fact, *khatd* could maintain a database of pending *khat* jobs, so that abuse of the system could be traced to the offending user.

2.4.5 Generating tickets for a user

When it is time for a batch job to run, the *khat* server must obtain a ticket for the user. It cannot simply issue a request for a ticket the way a user does because this request looks up the address from which it comes, and puts the client *IP* address into the ticket. Thus, the ticket must be constructed manually by the server. The steps taken by the server are as follows:

- Get the master key for Kerberos
- Get the TGT secret key from the Kerberos database
- Decrypt the TGT key with the master key
- Create a TGT ticket for the user
- Purge the master key and other keys from memory

A brief discussion of each of these steps follows. This discussion makes it clear that the *khat* server must have the Kerberos database available to it, and must either run on the same machine as Kerberos, or on a Kerberos slave machine.

2.4.5.1 Get the master key for Kerberos

The following steps are taken to get the master Kerberos key. We call *gettimeofday* to set the Kerberos time. Then we call routines to get and verify the

master key. After that, the version number is checked. If the wrong version number appears, or if *kdb_verify_master_key* returns an error, we log the error and exit. In this case, no tickets can be generated, and mail is sent to the user.

2.4.5.2 Get the TGT secret key from the Kerberos database

To get the TGT secret key from the Kerberos database, we call *check_princ*, which checks for expiration times on the master key and the service. It also populates the *Principal* data structure with information containing the key (encrypted under the master key), realm, *etc.*

2.4.5.3 Decrypt the TGT key with the master key

This step is straightforward. We use the master key to decrypt the TGT key.

2.4.5.3 Create a TGT ticket for the user

We have the TGT ticket that will be used to encrypt the ticket once it is constructed. Inside the ticket, we place user information such as name, instance, and realm, obtained from the call to *check_princ*. Then, we add the client host address that we obtained with a call to *getpeername* to see who the client was. In addition, we generate a random session key and include that in the ticket. Other usual information such as the time and ticket lifetime are also included. A lifetime of about 25.5 hours was chosen here, but that was arbitrary and based on the choice observed in existing Kerberos services. The ability to specify the lifetime could be added later as a user-specified option to *khat*.

2.4.5.4 Purge the master key and other keys from memory

Finally, we zero out all of the memory we used for highly sensitive information such as the master key. Even though this is a trusted machine, it never hurts to take added precautions.

2.4.5.5 Send the ticket to the client

Once the ticket has been constructed properly, it is encrypted under the session key available to the client and sent across the network. The client decrypts the ticket, and stores it with the batch job user as the owner, with no permissions for anyone else.

2.4.6 Renewing the tickets

If a batch job is to run for more than the lifetime of a TGT ticket, then the workstation must receive a new ticket for the user. The ability to renew tickets is a command line option. If a user prefers for jobs which outlive the tickets to die rather than have tickets generated until the job exits, he can chose to omit this option.

The *khat* program forks a master process to run the batch job. Before the job is actually run, this process forks a sub-process. This sub-process is in charge of maintaining the user's authentication on the workstation. When the master processing the job completes, it terminates the sub-process, and sends a message to the server that it is finished. This message is not necessary, but it is sent so that the *khatd* process on the server will exit normally. The details of this scheme follow.

The process that maintains the authentication works as follows. It sleeps until the authentication ticket is about to expire. Then, it calls *gettimeofday* to get the current time. The time is sent, encrypted, to the server, ensuring against replay, and proving possession of the secret key. The server checks the time, and if it matches to within a minute, is convinced that a new ticket must be sent. The server then constructs a new ticket for the user and sends it across the network. The workstation replaces its current ticket file with the new one, and then goes back to sleep.

The server on the other end waits for a request for tickets or a message from the client that the job has terminated. When a request comes in, the server checks that the time is within a minute of the current time. If not, the request is ignored, and no tickets are generated. If the time is correct, then the user creates a new TGT ticket for the user, encrypts it, sends it to the client, and continues waiting for requests.

2.5 KHAT runtime

A *khat* session consists of two phases. In phase one, the user invokes *khat*, and connects to a well known port on the server machine. When the spool file has been sent, the client and the server store away some information for future use and exit.

In phase two, it is time for a job to run, so the *khat* server connects to a well known port on the client machine, and the spool file and tickets are sent. This well known port is established by running a service, *khatrun*, on the client machine. Thus, the *khat* server becomes the *khatrun* client and the *khat* client becomes the *khatrun* server. For clarity, we will continue to use the terms client and server with respect to *khat*.

The server runs a program called *khatcheck* to determine when it is time to initiate phase two. *Khatcheck* runs in the background and wakes up every minute to see if there are any *khat* jobs to run. The name of the spool file contains the date and time for each job, so the current time is compared to the scheduled time for each job. If it is time, *khatcheck* forks and execs the *khatrun* program to begin phase two.

Khatrun needs some information for each job such as client name, instance, realm, and secret keys. *Khatd* stores this information in a file before it exits. The name of this file contains the name of the spool file for the job as a substring so it can easily be associated with the correct job. This is possible because *khatd* is assumed to be running on a secure machine.

2.6 KHAT Utilities

We have written a couple of Kerberos services to give *khat* users some flexibility. The commands *khatq* and *khatrm* were added to display a list of pending jobs, to display the contents of a pending job, and to remove a pending job.

When these commands are invoked, the user's workstation receives a *khat* ticket, and mutual authentication is performed. The *khat* server is then presented with this ticket along with the name of the service being requested. At this point, the user and the *khat* server have established a secure communication channel, as they are both in possession of a secret session key. The client and server use this session key to encrypt all message traffic between them for the remainder of their communication.

2.6.1 The KHATQ command

When the *khat* server receives a request for the *khatq* service from the client, it creates a list of all pending jobs for that user. A string is then created containing the time and date for the job to run. In addition, a unique job number is assigned to each job based on a hash function of the user's *uid* and the time and date for the job. Currently, a four digit number is assigned, but to insure a higher probability of uniqueness, larger numbers can be used.

The list of pending jobs is sent to the client. Since the length of this list may vary, a terminator string is sent to the client at the end of the list. Since each *khatq* request will cause a new session key to be established between the client and server, and as the terminator string is only used once per session, this string is not distinguishable from other data once it is encrypted.

If there are no jobs scheduled for the user, then only the terminating string is sent. The client then prints a message that there are no jobs scheduled. If the user specifies a job number as an option to the *khatq* command, then the server will send the contents of the batch job, as originally submitted, to the client, which will then print them on the standard output. If the user specifies a job number that does not exist, or does not belong to that user, *khatq* prints an error message. To prevent users from discovering job numbers of other users, this error message does not reveal any substantive diagnostic information.

One weakness of the current implementation of *khatq* is that traffic analysis could reveal some information. By analyzing the messages between the client and the server, an eavesdropper can discover how many jobs are pending or the approximate length of a batch job. This can be solved by padding all the messages to a given length, adding redundant messages, or some combination of the two.

2.6.2 The KHATRM command

When the *khat* server receives a request for the *khatrm* service, it verifies the ownership of the job, and then removes the spool file from the server machine. A message is sent to the client reporting success or failure. Again, a new session key is established for every invocation of *khatrm* so this message cannot be distinguished from other data. The client either prints that the job was removed, or that it could

not be removed. As in the *khatq* command, no further diagnostic information is printed so that it is impossible to discover if a job number represents a job scheduled by a different user.

2.7 Summary of KHAT

Using *khat* it is possible to run a batch job with authentication without manually renewing user tickets. The risks of having valid tickets on a vacant workstation are inherent to *khat*. To minimize the risk, the secret key used to authenticate to the server when the user is gone is maintained, encrypted in the process memory of the *khat* process on the client machine.

The *khat* server runs on a secure machine with access to the Kerberos database. It uses this database to generate tickets for users and the TGT service. Everything sent across the network is encrypted first with the secret key available on the workstation.

When scheduling jobs to run on a vacant workstation, there must be some security compromise for authentication to take place. This problem will remain until there is some way to reliably authenticate a workstation.

2.8 The Future of KHAT

We are considering some enhancements for future implementation. At present, *khat* can run jobs only on the workstation from which they were scheduled. It would be convenient if the first available workstation or a user-specified workstation could be used. One possibility for a user-specified workstation to run *khat* jobs is to require that the user log into the target workstation, authenticate to *khat*, and then leave an encrypted session key around. However, this is no different from the user simply logging into the target workstation and scheduling the job from there.

Another possible extension is to add flexibility to the *khat* service. For example, a user may wish to specify that a workstation should maintain valid tickets only between certain hours, when presumably, he believes it is safer. An option can be added in which the user specifies the lifetime of tickets, and perhaps provides some conditions under which they should or should not be renewed.

Another feature which could be added to *khat* is the ability for a process to save its state before a ticket expires, send it (encrypted) to the server, and then wait until the user reauthenticates to continue running. This feature would be useful in a case where the workstation (somehow) realizes it has been compromised, or is about to be.

Availability

The version of *khat* that is available contains a revision to the protocol presented in this chapter. This is due to a security flaw in the original version that was discovered using the techniques described in Chapter 4. A complete specification of the corrected protocol is found in Appendix D.

The code for *khat* can be obtained via anonymous FTP from */public/khat/khat.tar.Z* at *citi.umich.edu*. AFS users can find the file *khat.tar.Z* in the directory */afs/umich.edu/group/itd/citi/public/khat*.

Complete instructions for building and installing *khat* can also be found in the same directory in the file *khat.instructions*. The system is meant for use with AFS or with Version 4 of Kerberos, and assumes the availability of the *at* command (in binary form).

CHAPTER 3

FORMAL METHODS FOR AUTHENTICATION PROTOCOL ANALYSIS

3.1 Overview

In this chapter, we examine current approaches and the state of the art in the application of formal methods to the analysis of cryptographic protocols. We use Meadows' classification of analysis techniques into four types.

The Type I approach models and verifies a protocol using specification languages and verification tools not specifically developed for the analysis of cryptographic protocols. In the Type II approach, a protocol designer develops expert systems to create and examine different scenarios, from which he may draw conclusions about the security of the protocols being studied. The Type III approach models the requirements of a protocol family using logics developed specifically for the analysis of knowledge and belief. Finally, the Type IV approach develops a formal model based on the algebraic term-rewriting properties of cryptographic systems.

The majority of research and the most interesting results are in the Type III approach, including reasoning systems such as the BAN logic; we present these systems and compare their relative merits. While each approach has its benefits, no current method is able to provide a rigorous proof that a protocol is secure.

3.2 Introduction

Authentication is the process by which a principal in a distributed system proves its identity. Typically, each principal shares a secret with some trusted machine, called an authentication server. By proving possession of this secret, a principal

can establish trust in its identity. The use of passwords in a multi-user environment is an example of this.

The shared secret in an authentication system is typically used as an encryption key. The encryption scheme has the property that a user cannot generate or decrypt encrypted data without possession of the key. Thus, a principal proves it is in possession of a key by encrypting with it.

Authentication in a large, distributed system is challenging because principals communicate over a network that is vulnerable to many attacks. A passive intruder can eavesdrop on a line and obtain sensitive information. Of graver consequence, is an active intruder who can modify message traffic by blocking the transmission of packets and inserting his own packets at will. Such an intruder can impersonate any principal in the system and possibly intercept his rights and privileges.

Encryption can thwart the attacks of an active intruder. Encrypted data has the property that any modification to some part of the data causes the decryption to fail. Thus, without knowledge of the key, an active, malicious intruder's ability is limited to blocking data from reaching its destination.

In authentication systems, we assume that each principal shares a secret key with an authentication server. This key is established by some secure, off-line method. Two principals can communicate securely by sending encrypted messages to the authentication server, who can re-encrypt and forward them to the intended recipient. However, issues of scale make this impractical.

Rather, when two principals wish to communicate, they establish a secret key known only to them. This secret key serves as a secure communication channel between the two principals because an active intruder who doesn't know the key cannot successfully interfere with the communication[1]. However, establishing such a key, called a session key, is a nontrivial problem.

The problem of establishing secure session keys between pairs of principals in a distributed authentication system led to a great deal of research. This research focuses on the development of protocols, and is accompanied by a greater and more interesting problem, the analysis of authentication protocols.

---

[1]It is assumed that systems will always be vulnerable to message blocking because in the simple case an intruder can cut the physical wire connecting two machines.

The Needham and Schroeder authentication protocol [48] revolutionized security in distributed systems. Adaptations of this protocol, such as Kerberos [64] and the Andrew File System [28], have become universal. However, it was not long before a flaw was found in this protocol [17]. Needham and Schroeder then published a revised version of their protocol [49].

The existence of a subtle flaw in a previously trusted protocol stressed the need for formal methods for analyzing authentication protocols. In fact, many authors praise the merits of their analysis techniques with their ability to discover the flaw in the Needham and Schroeder protocol [9, 12, 23, 44, 63, 76].

A few specification techniques for authentication protocols have been published [41, 70, 77, 80], and several formal analysis techniques have been proposed. In particular, the use of predicate logic for the analysis of protocols was proposed by Burrows *et al.*[2] [9], and many extensions have since been published [11, 12, 21, 23, 62, 63].

Others have been critical of the BAN logic [50, 67], and have proposed their own logics [36, 39, 41, 42, 44, 46, 65, 67, 76, 80]. This chapter explores these logics and discusses the tradeoffs among them.

3.3 Terminology

This section describes some of the terminology used in the rest of this thesis. Because many researchers define their own terms and use different notations, we have standardized on the following definitions.

Threat model refers to the assumed characteristics of the security environment. It includes the assumptions made about the principals involved and the possible interference of malicious agents. Our threat model includes an active intruder who can delete, modify, and create message traffic at will. We also assume strong encryption.

Encryption is the science or art of generating a cipher text from a clear text, making the clear text unrecognizable. In security systems encryption involves the use of a secret key and a known algorithm.

[2]This logic is referred to as BAN logic, after the authors Burrows, Abadi, and Needham.

Decryption is the science or art of generating a clear text from a cipher text. In security systems decryption involves knowledge of a secret key and a known algorithm.

Cryptanalysis is the science or art of breaking a cryptographic code without knowledge of the key. The methods used take into consideration letter frequency, and any information about the context available. This type of analysis is very advanced and can defeat all but the best encryption techniques, but it may be computationally intensive.

Strong encryption is an encryption method that is assumed to be computationally unbreakable. Also, it is not vulnerable to any form of cryptanalysis.

Z is a common notation to represent the intruder. (It is also common to see X and C.)

Key management protocol is used interchangeably with the terms *authentication protocols* and *cryptographic protocols*. It is a set of rules defining the messages passed in an encryption system to distribute secret keys.

Nonce is an identifier, usually a large random number, that is used only once. The main purpose of a nonce is to link two messages together so that a response can be recognized as fresh. A nonce is usually represented as N_a or N_b, etc.

Doxastic logic is based on belief. The reasoning system uses rules about how belief is propagated to establish new beliefs.

Epistemic logic is based on knowledge. The reasoning is similar to reasoning in a doxastic logic, but these logics are used to reason about knowledge instead of belief.

$\{\boldsymbol{data}\}_k$ represents *data* encrypted under secret key, k.

Session key is a secret encryption key established between two principals for communication purposes. As the name implies, this key is intended for one session only. Sometimes this session is only one protocol run; often it lasts for the lifetime of a ticket or token.

Symmetric keys are used for private key systems. In such systems, the same key is used for encryption and decryption. For example, $\{data\}_k$ can be decrypted with k.

Asymmetric (public) keys are pairs of keys that are inverses of each other. One key is kept private, and is known only to the principal who possesses it. The other key is public, and is made widely available. Data encrypted with the private key can be decrypted with the public key; similarly, data encrypted with the public key can be decrypted with the private key.

Symmetric protocol exists if two principals play the same role in the protocol. Thus a protocol in which a principal speaks with the authentication server is not symmetric, whereas a protocol in which two users at the same trust level share data usually is symmetric.

3.4 Needham and Schroeder

We now turn to one of the most famous and landmark protocols to begin our discussion of protocol analysis.

The Needham and Schroeder protocol [48] distributes a secret session key between two principals in a network. The threat model of the Needham and Schroeder protocol assumes that each principal shares a secret key with an authentication server and that an intruder can read and modify anything that passes on the network. In addition, the model assumes that intruders can block any message from reaching its destination and insert malicious messages of their own.

The participants in this protocol are the three principals, A, B, and S, where S is the trusted authentication server, and A is a principal who wishes to initiate a secure session with principal B. Thus, as pointed out by Sidhu [60], this protocol is not symmetric. We represent a protocol step as

$$A \rightarrow B : Message$$

to indicate that A sends *Message* to B. Thus, the Needham and Schroeder protocol can be specified as follows:

1. $A \rightarrow S : A, B, N_a$

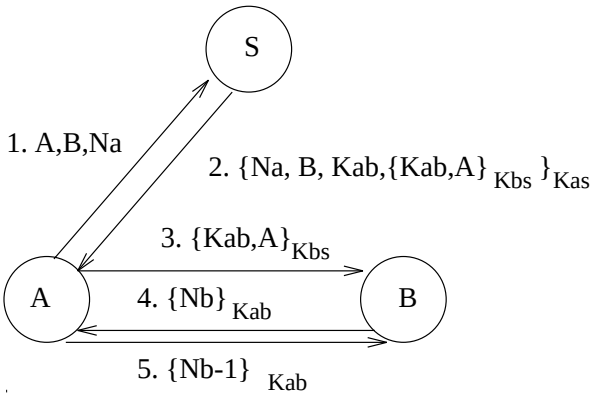

Figure 3.1: **The Needham and Schroeder Protocol**

2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$

3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$

4. $B \rightarrow A : \{N_b\}_{K_{ab}}$

5. $A \rightarrow B : \{N_b \Leftrightarrow 1\}_{K_{ab}}$

This protocol is represented graphically in Figure 3.1. Each node represents a principal, and the transitions represent the messages being sent. The transitions are numbered in the order of the messages. K_{ab} represents the secret key shared by A and B, etc.

In message 1, A sends a request to the server (S) indicating that it wishes to communicate with B. The nonce N_a is included to link future messages to this request. This message is sent in the clear because it includes no security related information.

In message 2, the server S responds with a session key, K_{ab}. A copy of the key is also encrypted under B's secret key. In addition, N_a is included as a guarantee that this message is not a replay of a previous response. Each principal is also told which principal will be on the other end of the secure channel. This can be seen by the inclusion of A in $\{K_{ab}, A\}_{K_{bs}}$.

In message 3, A forwards $\{K_{ab}, A\}_{K_{bs}}$ to B, who can decrypt it and recover K_{ab}. B then issues message 4 as a challenge to A to make sure that A possesses K_{ab}. In message 5, A proves possession of the session key. At the end of the protocol, it

would seem that A and B would be in possession of K_{ab},[3] and that no intruder could possibly know the secret session key. Thus, this protocol appears to allow A and B to establish a secure channel.

3.4.1 A Weakness in the Protocol

Denning and Sacco [17] were the first to discover a major weakness in the Needham and Schroeder protocol.

It is assumed that a session key is meant to be used only once and then discarded. Now, if we assume an intruder, Z, has recorded a previous run of the Needham and Schroeder protocol, then an attack is possible if the old session key is compromised.

To illustrate, suppose that an old session key, CK, has been compromised. If Z recorded the protocol run where CK was established, then Z can replay the message:

$$Z \rightarrow B : \{CK, A\}_{K_{bs}}$$

Thinking A has initiated a new conversation, B requests a handshake from A:

$$B \rightarrow A : \{N_b\}_{CK}$$

Z intercepts the message, decrypts it with CK, and impersonates A's response:

$$Z \rightarrow B : \{N_b \Leftrightarrow 1\}_{CK}$$

Thereafter, Z can send bogus messages to B that appear to be from A. B will have no way of knowing that it is not communicating with A.

3.4.2 Handling the Weakness

Denning and Sacco suggest that by adding timestamps to messages 2 and 3, the problem can be solved. Thus, these two steps become:

$$S \rightarrow A : \{T, N_a, B, K_{ab}, \{K_{ab}, A, T\}_{K_{bs}}\}_{K_{as}}$$

---

[3] We ignore the fact that S also has K_{ab} because it is assumed to be a trusted server that would not abuse the key.

$A \rightarrow B : \{K_{ab}, A, T\}_{K_{bs}}$

where T is a timestamp. Thus, a replay of message 3 would be recognized as old and would be ignored.

In a follow-up paper Needham and Schroeder propose a solution that is based on the use of nonces[49]. They observe that one of the communicating parties will require proof of the timeliness of a future message. It is always this party that should generate the nonce identifier.

This is achieved as follows. Before the protocol takes place,

$A \rightarrow B : A$

$B \rightarrow A : \{A, J\}_{K_{bs}}$, where J is a nonce identifier that will be kept by B.

Now, J can be included in the authenticator sent to A to be forwarded to B. Thus, B will be assured that the session key is fresh and not a replay.

The vulnerability of the Needham and Schroeder protocols comes from the fact that each session key is meant for exactly one session. If an intruder can compromise an old session key, he can force its use in another session. Both Denning and Sacco's solution and the revised Needham and Schroeder protocols solve this problem by requiring that the forwarded message from A to B establish a new session.

This section deals with symmetric secret keys. The arguments are similar for public key systems, and we do not repeat them here.

3.4.3 Discussion

We have shown how a weakness discovered in a published protocol can be fixed, but we have not proved that the resulting protocol is secure. Furthermore, we have not shown that a mechanical technique could discover the original weakness. In the remainder of this chapter we will discuss how formal methods have been applied to the analysis of authentication protocols.

3.5 Approaches to Analysis

Meadows [42] defines four approaches that have been taken in the analysis of cryptographic protocols:

Type I– To model and verify the protocol using specification languages and verification tools not specifically developed for the analysis of cryptographic protocols.

Type II– To develop expert systems that a protocol designer can use to develop and investigate different scenarios.

Type III– To model the requirements of a protocol family using logics developed for the analysis of knowledge and belief.

Type IV– To develop a formal model based on the algebraic term-rewriting properties of cryptographic systems.

The Type I approach is the least popular, while the Type III approach is the most common. These approaches share a few properties. In all cases, the methods are independent of the underlying cryptographic mechanisms.[4] In addition, we typically assume a set of principals and a trusted authentication server. The principals are not trusted, and may consist of a privileged intruder who can add, delete, or modify messages on the network at will.

The next four sections describe each of the four types of authentication protocol analysis. Table 3.1 shows the focus of current research. The entries in the table refer to the bibliography reference numbers.

3.6 Type I Approach

The Type I approach to the analysis of cryptographic protocols is to model and verify protocols using specification languages and verification tools not specifically developed for the analysis of such protocols. The main idea is to treat a cryptographic protocol as any other program and attempt to prove its correctness. A criticism of this approach is that it proves correctness and not necessarily security [60].

The first step in this approach is to specify the cryptographic protocol in a way that the techniques being used can be applied. Sidhu [60] suggests a specification technique that involves representing a protocol as a directed graph. Varadharajan [76] also adopts this method. However, in a more recent publication [77], he uses

[4]For a good discussion of failures in a cryptosystem due to the underlying encryption mechanisms see Moore [45].

LOTOS (Language of Temporal Ordering Specification) for specifying authentication protocols.

The work by Kemmerer [33] fits into several of the types of approaches, as shown in Table 3.1. The author describes an example system with a special cryptographic facility. The Type I approach can be seen in his attempt to use machine-aided verification techniques. The properties that the protocol should preserve are expressed as state invariants, and the theorems that must be proved to guarantee that the cryptographic facility satisfies the invariants are automatically generated by the verification system.

It should be noted that although much effort was concentrated on the Type I approach early on, most work in this area has been redirected as the logics of the Type III approach have gained popularity.

3.6.1 Using a Formal Verification System

Kemmerer [33] describes two goals in using formal methods for the analysis of encryption protocols. The first is to verify formally that an encryption protocol satisfies its stated security requirements, and the second is to discover weaknesses in its specification. His formal model uses a state machine approach where a system is viewed as being in various states, which are differentiated from one another by the values of state variables. The values of the variables can be changed only via well-defined state transitions.

Kemmerer uses an extension of first-order predicate calculus, a formal specification language called Ina Jo [57]. This nonprocedural assertion language was not developed specifically for use with security protocols, and thus this work fits into the Type I analysis approach.

Ina Jo uses the following symbols for logical operations:

& logical *AND*

$\rightarrow$ logical implication

In addition, there is a conditional form,

(if A then B else C)

where A is a predicate and B and C are well-formed terms. The notation for set operations is:

$\in$ is a member of

$\cup$ set union

{**a,b,...,c**} set consisting of elements a,b,...,and c

{**set description**} set described by set description

The language also contains the following quantifier notation:

$\forall$ for all

$\exists$ there exists

There are also two special Ina Jo symbols:

N'' to indicate the new value of a variable (e.g., $N''v1$ is the new value of variable $v1$)

T'' which defines a subtype of a given type, T

Kemmerer [33] describes an example system, and then gives an Ina Jo specification of the system. In this system, n terminals are connected to a central host. Each terminal contains a cryptographic facility that holds a permanent terminal key. The host stores two tables of keys. The first table is a list of the session keys being used in the system, and the second table contains the terminal keys. As such, the host acts as an authentication server.

In this system, the host is connected to a tamper-proof cryptographic facility that holds master keys for decrypting the information in the two tables. This system is used for pedagogical purposes and has not actually been implemented. The system architecture is shown in Figure 3.2[5]. Ina Jo *constants* and *variables* are described, along with *transforms*. An example of a constant in this example system is:

Terminal_key(Terminal_num):Key

[5]This figure is based on the figure by Kemmerer [33].

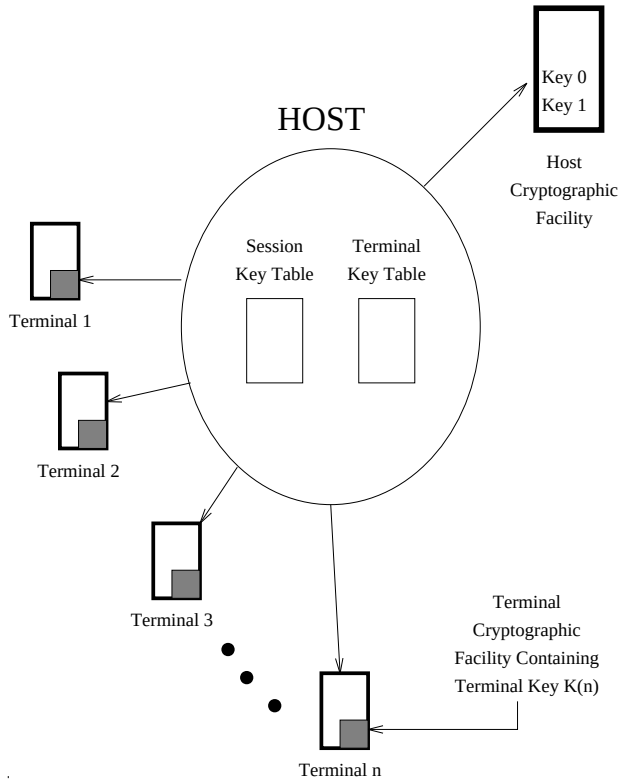

Figure 3.2: **System Architecture for Kemmerer's Sample System.**

because each terminal has a constant terminal key. However, as session keys vary from session to session, an example variable in Ina Jo is:

Session_Key(Terminal_num):Key

An example of a transform in Ina Jo is *Generate_Session_Key*. These are used to change state in the analysis.

An Ina Jo *axiom* is an expression of a property that is assumed. For example, to express that encryption and decryption are commutative, we would use the following Ina Jo axiom:

AXIOM $\forall t$:TEXT, $k1, k2$:Key (Encrypt($k1$, Decrypt($k2, t$)) =
Decrypt($k2$,Encrypt($k1, t$))).

Other such axioms are given in the full specification found in the appendix of Kemmerer's paper [33].

Finally, Ina Jo *criteria* clauses are used to specify the critical requirements that the system is to satisfy in all states. For example, the criterion that no key available to the intruder can be used for encryption can be specified as:

CRITERION $\forall k$:Key ($k \in$ Intruder_Info $\rightarrow k \notin$ Keys_Used).

Once the specification is complete, Ina Jo generates theorems that can be used to verify if the critical requirements (criterion) are satisfied. Kemmerer points out that "an advantage of expressing the system using formal notation and attempting to prove properties about the specification is that, if the generated theorems cannot be proved, the failed proofs often point to weaknesses in the system or to an incompleteness in the specification."

Kemmerer uncovers a weakness in his sample system using the formal specification. However, the value of this method is limited because proving the criterion of an Ina Jo specification does not necessarily guarantee that a protocol is secure. In addition, to specify requirements that secure a system from active attacks, the designer first needs to know the potential attacks, obviating any need for formal methods to discover them.

3.6.2 Using LOTOS for Protocol Specification

Varadharajan [77] proposes the use of LOTOS to analyze authentication protocols. He gives as examples the specification of two protocols that have been adopted as standards: the ISO/DP 9798 and CCITT X.509. However, no results are given. The paper concludes by stating that LOTOS tools are not yet adequate and are currently being developed.

The paper gives a very strong recommendation for the use of LOTOS. The goals of such a Formal Description Technique (FDT) are outlined as follows:

expressive power: ability to express a wide range of properties required for the description of services and protocols.

well-defined: syntax and semantics enabling mechanical manipulation, and validation.

well-structured: increasing understandability and maintainability of specifications.

abstraction: allowing representation of architectural aspects at a sufficiently high level of abstraction, where implementation details are not specified.

LOTOS has been developed for systems related to the Open Systems Interconnection (OSI), and is based on a process algebra that does not use a temporal logic, despite what the name might imply.

A system in LOTOS is modeled as a collection of processes in which the order of events is specified. As such, it can be used to model the messages sent in an authentication protocol. However, to date no concrete results have been reported using this method.

Methods of Type I will have to demonstrate some success before they become popular. The next section describes another attempt to use tools not originally intended to analyze authentication protocols.

3.6.3 Specifying a Protocol as a Finite State Machine

Sidhu [60] and Varadharajan [76] describe how to specify a protocol using state diagrams. A directed graph is used for each principal. First, an initial state is specified. Then, an arc is drawn to another state for each message that can be sent or received at that point. We will demonstrate this with an example.

The Needham and Schroeder [48] protocol is reproduced below for reference.

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b \Leftrightarrow 1\}_{K_{ab}}$

We use the following notation:

P^{-1} event – principal P transmits message 1

P^{+1} event – principal P receives message 1

Varadharajan [76] gives a state diagram for each entity, A, B, and S. The attempt is to capture the behavior of each principal in the protocol. However, his example

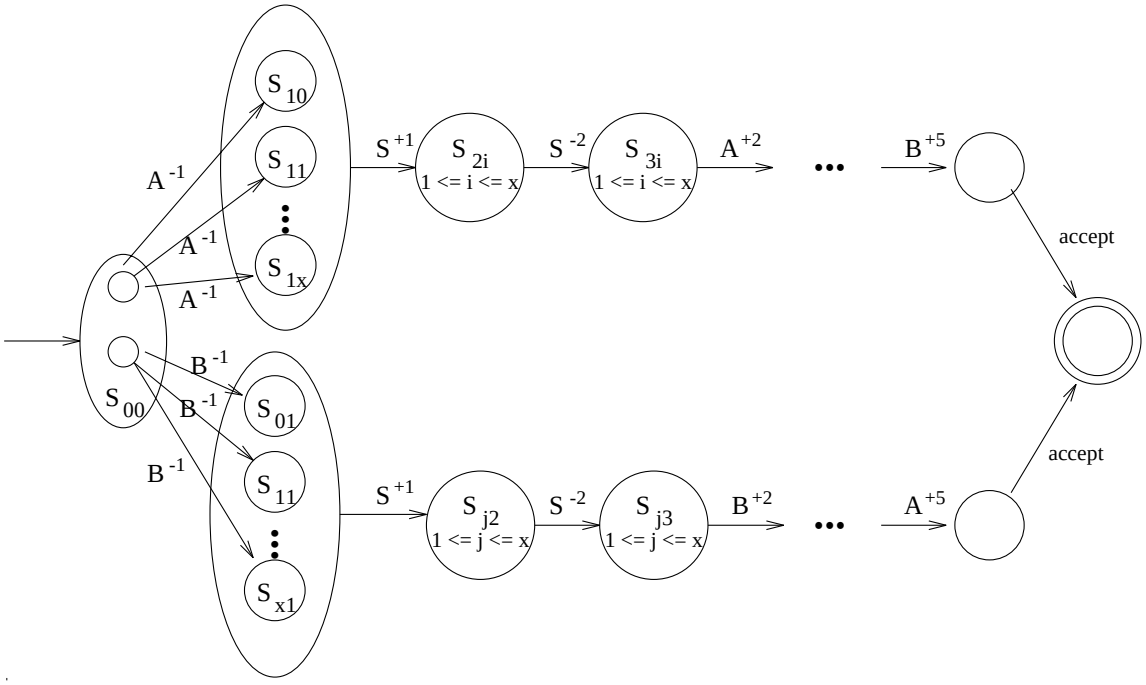

Figure 3.3: **Nondeterministic Finite State Machine for Principals A and B Initiating the Needham and Schroeder Protocol.** The arc P^{-n} means that principal P transmits message number n. P^{+n} means that P receives message n. This machine is constructed by taking the cross product of the individual machines for A and B initiating the protocol. If a state of A's machine is S_i, and B's is labeled S_j, then the corresponding state in the cross product machine is S_{ij}. The number of legal states in each of A's and B's machines is x, and the cross product contains $x^2 + 1$ legal states including the accepting final state. All other states are illegal, and stand for illegal runs of the protocol.

is highly complex and counterintuitive. We prefer to represent the protocol as a cross product of the state diagrams for each individual principal (Figure 3.3). The nondeterministic finite state machine is constructed from the individual machines for A and B. The individual machine for a principal is composed of a sequence of states with arcs representing the transmission or reception of a message. A state is labeled P^{-n} to indicate that principal, P has transmitted message number n and P^{+n} if P receives message n.

If the final accepting state is reached, then we have a legal run of the protocol initiated by either A or B. If an individual principal's machine consists of x states, then the cross product machine with another principal in the protocol has $x^2 + 1$ states including the final accepting states. All other states represent illegal runs of the protocol.

As we describe each state in our protocol specification, notice that it is assumed that A and B play the same role in the protocol. This assumption is controversial. Varadharajan [76] states that "A and B have symmetric roles." However, Sidhu [60] states that "The authentication protocols of Needham and Schroeder are not symmetric between a sender and a receiver and assume a particular time ordering of events."

The state representations presented by Sidhu differ slightly from those of Varadharajan. Remaining impartial, we present our own state diagram construction, and refer the curious reader to Sidhu's and Varadharajan's papers [60, 76] for their representations.

The next section discusses how these finite state machines that are used to specify protocols can also be used in their analysis.

3.6.4 The Use of Finite State Machines for Protocol Analysis

The state machines described above can be used to analyze authentication protocols by employing a technique known as the reachability analysis technique [79].

To use this technique, for each transition, the global state of the system is expressed using the states of the entities and the states of the communication channels between them. Each global state is then analyzed and properties are determined, such as deadlock and correctness. If an entity is not able to receive a message that it is supposed to receive in a given state, then there is a problem with the protocol. For an example of such an analysis, see Varadharajan [76].

Reachability analysis techniques are effective in determining whether or not a protocol is correct with respect to its specifications, the purpose for which they were invented. However, they do not guarantee security from an active intruder. The weakness of Type I analysis techniques is that in applying methods that were not intended specifically for security analysis, subtle pitfalls that are peculiar to the security domain, such as the effect of message replay, are not considered.

3.7 Type II Approach

The Type II approach to the analysis of cryptographic protocols is to develop expert systems that a protocol designer can use to develop and investigate

different scenarios. These systems begin with an undesirable state and attempt to discover if this state is reachable from an initial state.

Although this approach may better identify flaws than Type I approaches, it does not guarantee the security of an authentication protocol, nor does it provide an automated technique for developing attacks on a protocol. In other words, the Type II approaches can discover whether a given protocol contains a given flaw, but are unlikely to discover unknown types of flaws in protocols.

Longley and Rigby [36] summarize the value of expert systems in the analysis of key management schemes. The expert systems provide:

- a new perspective on an authentication system;
- a technique of building models capable of continuous refinement;
- a method of interaction with the model, which provides a greater insight into the operation of the system;
- a model that responds to *what if* questions; and
- a method of testing the effects of proposed system modifications.

Thus, expert systems can be used in conjunction with other analysis techniques such as those of Type III and IV for the purposes mentioned above, but they will never replace those techniques.

The NRL protocol analyzer [70] might be viewed as a Type II approach. However, because it is based on the Dolev and Yao model [19], in which an intruder produces words in a term-rewriting system, we will consider it a Type IV approach.

3.7.1 The Interrogator

The Interrogator, by Millen *et al.* [44] is a noteworthy effort to apply expert systems to the analysis of security protocols. The input to the system is a protocol specification and a target data item. The output is a message history showing how the penetrator could have obtained this data item.

In the Interrogator, a protocol is viewed as a collection of communicating processes, one for each principal. Each process has a set of possible states, and the transmission of a message can cause a state transition in a process. Each process

maintains its own state, and when applicable, sends messages to other processes causing them to change state. The system is based on the finite state machine approach [27].

The Interrogator generates a large number of paths through a protocol, ending in a specified insecure state. If any of these paths start with an initial state, then a vulnerability has been discovered. Thus, an important issue in using the Interrogator is the specification of the final state.

In the Interrogator, the penetrator is expressed as a relation:

$\texttt{p\_knows}(x, H, q)$

where x is the data item learned by the penetrator, H is the message history that lead to this discovery, and q is a state of the network reachable from the initial state. The meaning of `p_knows` is as follows:

$\texttt{p\_knows}(x, H, q)$ **iff**

x is known initially

or $(H = H'\text{sent}(m)$ **and** $\text{sent}(m) : q' \rightarrow q$ **and** $H' : q_0 \rightarrow q'$ **and** $\texttt{p\_gets}(x, m, H', q'))$

or $(H = H'e$ **and** $e : q' \rightarrow q$ **and** $\texttt{p\_knows}(x, H', q'))$

or $(H : q_0 \rightarrow q'$ **and** $\texttt{p\_modifies}(q', q, H)$ **and** $\texttt{p\_knows}(x, H, q'))$

Similarly, `p_gets` is defined as follows:

$\texttt{p\_gets}(x, m, H, q)$ **iff**

x is a field of m

or $(\{m'\}_k$ is a field of m **and** $\texttt{p\_knows}(k, H, q)$ **and** $\texttt{p\_gets}(x, m', H, q))$

The definition of `p_knows` describes the three ways a penetrator may learn x with message history, H, in state q. The penetrator may learn it from the last message read; may have already known it in the previous network state, q'; or may learn it using `p_modifies` described below.

The definition of `p_gets` states that a penetrator can read any message, but if some part of the message is encrypted, then it can only be extracted if the key encrypting that field is known. The statement $\texttt{p\_modifies}(q', q, H)$, describes how a penetrator who modifies the network buffer can learn x. Millen *et al.* state that

> "`p_modifies`(q', q, H) is characterized by saying that if m is a new message in the network buffer of the new state q, the penetrator knows each field of m in the prior state q' reached by history H." [44]

This means that if the penetrator knows x in state q', reachable with message history H, and the penetrator changes the message buffer such that state q is reached instead with message history H, the penetrator still knows x.

Millen *et al.* claim that the Interrogator was able to rediscover the flaw in the Needham and Schroeder protocol. However, the Interrogator was first provided with information to the effect that the penetrator knows an old connection key. This information could be supplied because the programmers of the Interrogator were familiar with the weakness in the Needham and Schroeder protocol.

Systems such as the Interrogator can be useful for providing message histories for *known* attacks, but it remains to be seen whether such methods will discover new attacks on protocols previously believed to be secure. No such result has been reported.

3.7.2 A Rule-Based System

Longley and Rigby [36] describe a rule-based system used to test the vulnerability of a key management scheme to specified attacks. The results of applying this system to the IBM key management scheme described by Davies and Price [16] were consistent with the known characteristics of that scheme.

The expert system uses an exhaustive search to determine if a given attack is successful. When the system halts, then the history of rule firings can give an attack strategy. Until then, nothing can be said about the given attack. In fact, in some cases, the search space is infinite and the system does not even halt.

Longley *et al.* use a rule-based system, OPS5 [7]. This system uses rules to transform goals into sub-goals, and this process is continually refined until a concrete attack strategy is reached.

At best, this system can be used as a model of threat analysis. It does not perform the function of analysis in terms of demonstrating the security of an authentication protocol. Rather, it can *sometimes* determine how a given attack might be successful against a protocol.

3.7.3 Discussion

The Type II approaches to protocol analysis serve a limited function. They are most useful for analyzing known weaknesses in protocols, and generating message lists to exploit those weaknesses.

The systems developed under this approach are usually inefficient, often resorting to exhaustive search. In addition, the results are often inconclusive, and the systems may not even halt.

Their limitations are due to the lack of expressiveness of the types of rules found in expert systems. For this reason, the majority of research into the analysis of authentication protocols falls into the Type III category, discussed next.

3.8 Type III Approach

The Type III approach to protocol analysis uses formal logic models developed for the analysis of knowledge and belief. Burrows *et al.*'s landmark BAN logic [9] initiated intense research using this approach. Since then, BAN has been extended [11, 12, 21, 23, 38, 62], and criticized [38, 50, 62, 67].

This section discusses other contributions to the Type III approach including the logic of Bieber [4] and its extension by Snekkenes [63]; the axiomization of trust and belief by Rangan [52]; the logic of Syverson [65]; the logic of Kailar *et al.* [31]; and the logic of Moser [46].

In addition to these logics, some work has concentrated on the semantics of logics for authentication protocols [2, 71]. We discuss the semantics introduced here and why it is important to define the semantics of a logic with great care.

In the following sections we present these logics, discuss, compare, and evaluate their relative merits.

3.8.1 An Axiomization of Belief

Much of the work in the Type III approach is based on a formal axiomization of belief and trust. Shoham and Moses [59] describe the relationship between knowledge and belief, and note a close connection between belief and nonmonotonic reasoning.

Syverson [71] shows that belief and knowledge are equally adequate for protocol analysis on the logical level. Logics based on knowledge are termed epistemic, while doxastic logics refer to those based on belief. The main difference in reasoning with these two logics is that all epistemic logics have an axiom that states that if a principal knows X, then X. No doxastic logics have such an axiom. However, Syverson shows that this axiom (termed axiom **T**) can easily be captured in doxastic logics.

Rangan [52] provides an axiom schema for belief that is frequently referenced in the literature. In his notation, the term $B_i p$ means that principal i believes p. The schema is as follows.

for all $i, i = 1, ..., m$:

A1 All substitution instances of propositional tautologies.

A2 $B_i p \wedge B_i(p \Rightarrow q) \Rightarrow B_i q$.

A3 $B_i p \Rightarrow B_i B_i p$ (introspection of positive belief).

A4 $\neg B_i p \Rightarrow B_i \neg B_i p$ (introspection of negative belief).

A5 $\neg B_i$(false) (process i does not believe a contradiction).

The following are the inference rules.

for all $i, i = 1, ..., m$:

R1 From p and $p \Rightarrow q$ infer q (modus ponens).

R2 From p infer $B_i p$ (generalization).

In his paper [52], Rangan defines the *transitivity, Euclidian,* and *serial* properties, and shows that **A3** corresponds to transitivity, given **R2**, **A4** corresponds to the euclidian property, and **A5** corresponds to the serial property.

These definitions of knowledge and belief are the foundation for the Type III approaches discussed below.

3.8.2 The BAN Logic

The BAN logic casts authentication protocols in formal terms to reason about the state of belief among principals in a system. The authors' goals were to be able to answer the following questions about a protocol:

- What does this protocol achieve?
- Does this protocol need more assumptions than another one?
- Does this protocol take any unnecessary steps, ones that could be left out without weakening it?
- Does this protocol encrypt a message that could be sent in the clear without weakening security?

The authors state that such issues as errors introduced by concrete implementations of a protocol, such as deadlocks, or inappropriate use of a cryptosystem (as described by Voydock and Kent [78]) are not considered; this system deals with authentication protocols on an abstract level only.

3.8.2.1 The basic constructs of the BAN logic

The only propositional connective is conjunction, which is denoted with a comma. Associativity and commutativity properties are taken for granted.

P *believes* X The principal, P, acts as though X is true.

P *sees* X Someone has sent a message containing X to P, who can read and repeat X (possibly after doing some decryption).

P *said* X At some time, the principal P sent a message that includes the statement X. It is not known how long ago the message was sent, or even if it was sent during the current run of the protocol. It is known that P believed X when he said it.

P *controls* X The principal P is an authority on X and should be trusted on this matter. This construct is used primarily when a principal has delegated authority over some statement.

#(X) The formula X is *fresh*. That is, X has not been sent in a message at any time before the current run of the protocol. This is defined to be true for *nonces*, that is, expressions generated for the purpose of being fresh.

$P \overset{K}{\leftrightarrow} Q$ P and Q may use the *shared key* K to communicate. The key K is good, in that it will never be discovered by any principal except P or Q, or a principal trusted by either P or Q.

$\overset{K}{\mapsto} P$ P has K as a *public key*. The matching *secret key*, K^{-1}, will never be discovered by any principal other than P or a principal trusted by P.

$P \overset{X}{=} Q$ The formula X is a secret known only to P and Q, and those principals to whom they reveal it. P and Q may use X to prove their identities to one another.

$\{X\}_K$ from P This represents the formula X encrypted under the key K by principal P. The from P part is often omitted, and it is assumed that each principal is able to recognize and ignore his own messages.

Logical postulates are formed from these basic constructs. A security protocol is idealized, according to rules defined by the authors, in terms of these postulates. Every protocol must be idealized before using the BAN logic; many examples follow.

3.8.2.2 The rules of inference of the BAN logic

Burrows *et al.* [9] provide rules of inference for reasoning about the belief in a protocol. These rules are applied to the initial assumptions to drive a proof or to answer questions about a protocol. One important rule, the *message meaning rule*, states how to derive belief from the origin of a message.

$$\frac{\text{P believes Q} \overset{K}{\leftrightarrow} \text{P, P sees } \{X\}_K}{\text{P believes Q said X}}$$

Remember that $\{X\}_K$ in this context stands for $\{X\}_K$ from $R \neq P$. Then this formula can be intuitively explained as:

> IF P believes that Q and P share a secret key, K, and P sees X, encrypted under K, and P did not encrypt X under K, THEN P believes that Q once said X.

A similar postulate exists for public keys and shared secrets. Another important rule of inference for the BAN logic is the *nonce-verification rule*.

$$\frac{\text{P believes } \#(X),\ \text{P believes Q said X}}{\text{P believes Q believes X}}$$

For the sake of simplicity, the authors of BAN state that X must be clear text, that is, it should not include any subformula of the form $\{Y\}_K$. An intuitive explanation of this rule is:

> IF P believes that X could have been uttered only recently and that Q once said X, THEN P believes that Q believes X.

This rule is important because many protocols rely on the use of nonces to avoid successful replay attacks. In fact, this is the only postulate that promotes from *said* to *believes*, and thus reflects in an abstract way, the practice of using challenges and responses for authentication. A result of applying this rule demonstrates that challenges often need not be encrypted, but responses must be.

The next rule, the *jurisdiction rule*, is often used for delegation.

$$\frac{\text{P believes Q controls X, P believes Q believes X}}{\text{P believes X}}$$

This rule states:

> IF P believes that Q has jurisdiction over X, and P believes that Q believes X, THEN P believes X.

Burrows *et al.* [9] provide many other inference rules that can be used to combine beliefs.

3.8.2.3 The idealized protocol of the BAN logic

For a protocol to be analyzed using the BAN logic, it must first be converted to an idealized form. Typically, a step in a protocol is written as:

$$\text{P} \rightarrow \text{Q} : \text{message}$$

This means that P sends the message and that the principal Q receives it. This framework is often ambiguous, and does not lend itself to formal analysis. For example, when something is encrypted under a session key, it may not always be clear what

parts of the message are fresh, or who exactly knows this key. Therefore, each step in a protocol is transformed into an idealized form. A message in the idealized protocol is a formula. Say we define the protocol step:

$$A \rightarrow B : \{A, K_{ab}\}_{K_{bs}}$$

In this step, A tells B, who knows the key, K_{bs}, that K_{ab} is a key to communicate with A. It is clear that A did not generate this message, because A does not know K_{bs}. In fact, the message must have come from the server S. This step is idealized as:

$$A \rightarrow B : \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$$

When this message is sent to B, we can deduce that the formula

$$\text{B sees } \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{bs}}$$

holds, indicating that the receiving principal becomes aware of the message and can act upon it.

In the idealized form, parts of the message that do not contribute to the beliefs of the recipient are omitted. Thus, clear text parts of the message are not included, because they can be intercepted and read or forged by anyone. Idealized messages are of the form $\{X_1\}_{K_1}$,...,$\{X_n\}_{K_n}$, where each encrypted part is treated separately.

The authors of BAN logic "view the idealized protocols as clearer and more complete specifications than the traditional descriptions found in the literature, which we view merely as implementation-dependent encodings of the protocols" [9]. However, no clear transformation method is presented. The paper gives numerous examples of the transformation to an idealized protocol; after careful study, idealizing protocols becomes intuitive. However, Woo and Lam [80] criticize the idealization of protocols. "We find idealization undesirable because of the potentially large semantic gap that exists between the original protocol and the idealized version."

Nessett's criticism [50] raises similar concerns, as we shall see.

3.8.2.4 Protocol analysis with the BAN logic

The steps in protocol analysis with BAN logic as presented by its authors are:

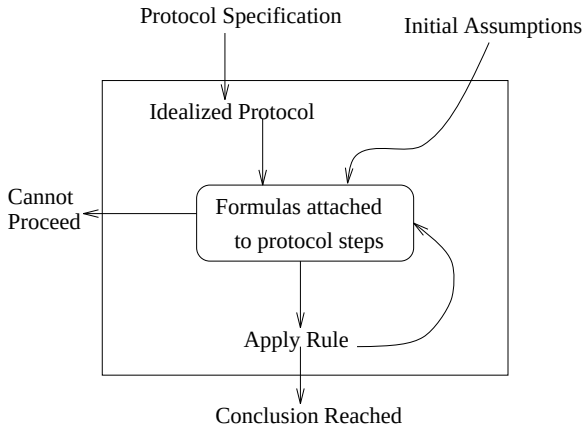

Figure 3.4: **Protocol Analysis with the BAN Logic:** The input to BAN is a protocol specification and the initial assumptions. At each step, formulas are attached to the protocol messages, and either a rule is applied, or the logic must halt. If possible, the desired conclusion is reached.

1. The idealized protocol is derived from the original one.

2. Assumptions about the initial state are written.

3. Logical formulas are attached to the statements of the protocol, as assertions about the state of the system after each statement.

4. The logical postulates are applied to the assumptions and the assertions to discover the beliefs held by the parties in the protocol.

More precisely, a protocol in the BAN logic is an ordered series of "send" statements, S_1,...,S_n, each of the form $\mathrm{P} \rightarrow \mathrm{Q} : \mathrm{X}$ with $\mathrm{P} \neq \mathrm{Q}$. An *annotation* for a protocol consists of a sequence of assertions inserted before the first statement and after each statement. The assertions are made by combining formulas of the forms P *believes* X and P *sees* X. The first assertion contains the assumptions, while the last assertion contains the conclusions. These are similar to simple formulas in Hoare logic [26]. They are written in the form:

$$[\text{assumptions}]$$
$$S_1\ [\text{assertion 1}]\ S_2 \ldots [\text{assertion } n \Leftrightarrow 1]\ S_n$$
$$[\text{conclusions}]$$

Protocol analysis with the BAN logic is summarized in Figure 3.4. The protocols

use no notion of time. Instead, time is divided into past and present depending on whether something was said in a previous or current run of the protocol.

The authors of BAN state that "More ambitious proofs may require finer temporal distinctions, reflected by constructs to reason about additional epochs, or even general-purpose temporal operators (see, for example, Halpern & Vardi 1986 [24])." In a recent paper, Syverson [68] introduces temporal axioms to the BAN logic and exposes protocol flaws using this extended logic.

3.8.2.5 The goals of authentication of the BAN logic

There is some debate as to what the goals of authentication are. Some argue that authentication is complete between A and B if there is a K such that:

$$\text{A believes A} \overset{K}{\leftrightarrow} \text{B}$$

$$\text{B believes A} \overset{K}{\leftrightarrow} \text{B}$$

Others believe that an authentication protocol should achieve:

$$\text{A believes B believes A} \overset{K}{\leftrightarrow} \text{B}$$

$$\text{B believes A believes A} \overset{K}{\leftrightarrow} \text{B}$$

The first set of goals is referred to as first-level belief, whereas the second set is termed second-level belief. According to Syverson [67], the level of belief needed varies for different applications and should be specified along with the protocol; the goals of BAN logic have often been misinterpreted.

Cheng and Gligor [14] claim the following conditions for the BAN logic must be satisfied at the end of a protocol run:

1. Both A and B believe K_{ab} is a secret key shared exclusively between A and B.

2. Both A and B believe that the other has the first-level belief. This is the second-level belief. If a party holds a second-level belief, then it believes that a secure channel has been established.

3. The causal relation between the first-level and second-level belief holds. That is the first level-belief must be established at some time before the second-level belief.

4. K_{ab} should be distributed exclusively to A and B; thus no parties other than A and B should have beliefs about K_{ab} ([14], p. 222).

Syverson [67] claims that these goals contradict the original goals set out by Burrows *et al.* In particular, using BAN logic, any principal who A and B trust can also be delegated the key, K_{ab}. Also, as Syverson points out, a second-level belief is not mandated by Burrows *et al.*, as Cheng and Gligor claim.

3.8.2.6 Nessett's criticism of the BAN logic

The BAN logic has been successful in finding flaws in some well known protocols, such as the Needham-Schroeder protocol, the Andrew secure RPC handshake, and the CCITT X.509 protocol. In addition, BAN has uncovered redundancy in the Needham-Schroeder conventional key protocol, the Otway-Rees protocol, Kerberos, the Yahalom protocol, the Andrew RPC handshake, and the CCITT X.509 protocol. As such, BAN logic can be called a success.

However, there are some problems with BAN logic. One problem is pointed out by Nessett [50], who demonstrates what he claims to be the hazards of devising systems of logic. He states that "a simple example shows that the BAN logic is capable of deducing characteristics about security protocols that by inspection are obviously false."

In Nessett's example, two principals, A and B communicate using public keys. The protocol is:

$$A \rightarrow B : \{N_a, K_{ab}\}_{K_a{}^{-1}}$$

$$B \rightarrow A : \{N_b\}_{K_{ab}}$$

The idealized form presented by Nessett is:

$$A \rightarrow B : \{N_a, A \overset{K_{ab}}{\leftrightarrow} B\}_{K_a{}^{-1}}$$

$$B \rightarrow A : \{A \overset{K_{ab}}{\leftrightarrow} B\}_{K_{ab}}$$

A sends B a message, containing K_{ab}, the secret key between A and B, encrypted under A's private key. Thus, as the corresponding public key is well known, the key K_{ab} is no longer a secret. In the example, B then responds with a nonce identifier

N_b, encrypted under the shared key, K_{ab}. According to the BAN model, this nonce is fresh and secret. However, as Nessett points out, it is obvious that N_a is readable and forgeable by anyone.

The problem with the BAN logic is that there is no way to represent what a principal does *not* know. All of the constructs and postulates deal with what a principal *does* believe, but there is no way to represent that a principal cannot know something. As Nessett states [50] "The essence of this flaw rests in the inability of the logic to analyze security protocols to assure that private information remains private."

Burrows *et al.* [10] defend their logic. They claim that the main difficulty in BAN logic as pointed out by Nessett is the assumption that A believes K_{ab} is a good shared key for A and B. "This assumption is clearly inconsistent with the message exchange, where A publishes K_{ab}. The inconsistency is not manifested by our formalism, but is not beyond the wit of man to notice."

Syverson [67] states that the confusion arises because "the BAN logic deals only with trust and not with security." Thus, he claims that Nessett's criticism is not valid because BAN does not claim to provide security, but rather, trust.

On the other hand, Snekkenes [62] attributes the Nessett flaw to the BAN logic being restricted to partial correctness. He defines a class of protocols called *terminating*, and shows that the Nessett protocol is a non-terminating one. "A statement or protocol step S terminates after finite time only if FALSE is not a derivable assertion succeeding S."

The criticism by Nessett has sparked a debate that has led to a clearer understanding of the role of knowledge and belief in the analysis of key management schemes. Much work has referred back to the research of Shoham and Moses [59] that specifically defines the relationship between belief and knowledge.

3.8.3 Extensions to the BAN Logic

The BAN logic was purposely designed to be open ended. That is, new constructs and postulates can be added to suit a particular application. It is not unusual to customize the BAN logic for an application to analyze a protocol to which the original BAN logic does not directly apply.

Some of these extensions have focused on eliminating some of the assumptions in the original BAN logic. Others have been necessary for expanding the reasoning power of BAN. In the following sections we discuss some extensions to the logic.

3.8.3.1 The GNY logic

The Gong, Needham and Yahalom [23] extensions to the BAN logic are often referred to as the GNY logic. Gong *et al.* describe new constructs that eliminate some of the assumptions made by the original BAN logic. In particular, the GNY logic does not assume that redundancy exists in encrypted messages. Instead, they introduce the notion of *recognizability* to represent the fact that a principal expects certain formats in the messages it receives. Also, Gong *et al.* explicitly represent whether a principal generated a message itself.

The notion of *recognizability* is important. A principal participating in a protocol has expectations about the messages he will receive, and the analysis technique should take these into consideration. Thus, if a protocol step specifies that A will receive nonces, N_a and N_b, then the next two values received will be treated as nonces. Logical postulates are added to require that a principal's expectations according to the protocol are met. These rules are defined below.

One of the important contributions of the GNY logic is the recognition that belief and possession are different. In this extended logic, each principal maintains a *belief set* and a *possession set.* Along with the basic constructs of BAN, the following are included in the GNY logic:

P $\triangleleft$ X P is told formula X. P receives X, possibly after performing some computation such as decryption. A formula being told can be the message itself, as well as any computable content of that message.

P $\ni$ X P possesses, or is capable of possessing formula X. At a particular stage of a run, this includes all the formulae that P has been told, started the session with, or was able to compute for formulae he already possesses.

ϕ(x) The formula X is recognizable. If P *believes* ϕ(x), then P would recognize X if P had certain expectations about the nature of X.

P $\propto$ **X** P is eligible to send formula X. A principal is only eligible to send something that he possesses or can construct. P is eligible to send formula X. A principal is only eligible to send something that he possesses or can construct.

A formula in GNY may also be regarded as a *not-originated-here* formula, meaning that it was not previously generated by a principal in the current run. This is represented by adding an asterisk (*) to the formula; and the paper [23] describes a mechanical process by which this is achieved.

The following postulates are defined:

$$\frac{\mathrm{P} \triangleleft \mathrm{X}}{P \ni X}$$

which states that principals possess what they are told, and

$$\frac{P \ni X, P \ni Y}{P \ni (X,Y), P \ni F(X,Y)}$$

which states that if a principal possesses X and Y, he also possesses the concatenation of X and Y, and any computable function F of X and Y. Similarly for recognizability,

$$\frac{\mathrm{P}\ \textit{believes}\ \phi(\mathrm{X})}{P\ \textit{believes}\ \phi(X,Y), P\ \textit{believes}\ \phi(F(X))}$$

states that if a principal believes that X is recognizable, then that principal believes that the concatenation of X with anything is recognizable, and that the application of some computable function to X is recognizable.

An important postulate in GNY states that if

$$\frac{C1}{C2}$$

is a postulate, then for any principal, P, so is

$$\frac{P\ \textit{believes}\ C1}{P\ \textit{believes}\ C2}$$

This is called the rationality rule and allows principals to reason about the state of other principals.

Gong *et al.* define preconditions that can be attached to rules to achieve different levels of belief. "Since we do not require the universal assumption that all principals are honest and competent, we should reason about beliefs held by others based on trust of different levels." Precondition statements are attached to formulas, and GNY provides trust and jurisdiction postulates for reasoning with preconditions.

The GNY logic is used to uncover the weakness in the Needham and Schroeder protocol. Then, the enhanced Needham and Schroeder protocol is analyzed and a second-level belief is attained. This logic is an improvement over the BAN logic in that it separates the content from the meaning of a message. Thus, the results of an analysis will be determined by the level of trust placed between principals. The original BAN logic did not accommodate for different levels of trust. Thus, the GNY logic increases the classes of protocols that can be analyzed.

In a later paper, Gong describes enhancements to the GNY logic to handle infeasible protocol specifications [22]. The problem is that a specification that could not possibly represent a real world situation can still be verified to be correct in the BAN and GNY logics. An example of such an infeasible specification is a protocol in which P sends R's password to Q. If R's password is represented by X, then this protocol step is:

$$P \rightarrow Q: \ \{X\}_{K_{PQ}}$$

As P and Q are not supposed to know R's password, this protocol is not feasible, and yet the GNY logic could not detect this.

Another type of infeasible specification that GNY and BAN cannot detect can lead to beliefs that do not preserve a causal relation. Say we have the protocol step:

$$P \rightarrow Q: \ \{P \ believes \ P \overset{S}{\leftrightarrow} Q\}_{K_{PQ}}$$

This will cause Q *believes* P *believes* $P \overset{S}{\leftrightarrow} Q$, however, if it is not the case that the statement P *believes* $P \overset{S}{\leftrightarrow} Q$ already existed, then the causal relation between beliefs is not preserved. Without guarantees of causality, part of a causal chain may be broken, and then the path may not be trusted (e.g. [31]). The causal chain is broken any time a principal, P sends a message that contains a belief, and P does not hold that belief.

The BAN logic assumes that principals believe what they say, and so the infeasible specification due to causal relations is not an issue. Gong introduces the notion of *eligibility* to the GNY logic. Thus, if $P \propto X$, then P is eligible to send formula X. Thus, new postulates are added to the logic:

$$\frac{P \rightarrow Q: \ X, P \propto X}{Q \triangleleft X}$$

This rule states that if P sends X to Q, and P is eligible to send X, then Q receives X. Similarly,

$$\frac{P \ni X}{P \propto X}$$

This rule says that if P possesses X, then P is eligible to send X. Other rules are included to describe when a principal is eligible to encrypt with a key, K, or to perform a hash function.

Thus, using the extensions to the GNY logic, we can reason about protocols whose specification fall into two categories of infeasibility. Although Gong's work is useful, it appears that a more formal technique is needed to discover infeasible protocols in the general case.

3.8.3.2 The Mao and Boyd logic

Mao and Boyd [38] describe four weaknesses in the BAN logic and propose a new logic, based on BAN, which offers several improvements. Mao *et al.* also present a fault in a version of the Otway-Rees protocol [51] that the authors of the BAN logic proved to be correct [9]. Finally, a revised version of Otway-Rees is presented by Mao *et al.* and proved correct with the new logic.

Mao and Boyd discuss the following defect of the BAN logic:

1. Protocol Idealization
2. Belief
3. Protocol Assumptions
4. Confidentiality

The authors discuss these defects and give examples.

Protocol idealization suffers from too much flexibility. New terms and constructs can be freely chosen from an infinite alphabet provided by the original BAN logic. Also, as Mao *et al.* point out, "There seems to be no well-understood semantic rule to govern this job of idealization." [38]

The flaw in protocol idealization can be seen clearly if we view a protocol specification as a procedure declaration. Once formal parameters are substituted with "real values," the behavior is unpredictable. The authors make this analogy and and

use it to demonstrate the flaw in the Otway-Rees protocol. The GNY logic [23] is also criticized for using the same idealization rules, and thus suffering the same weakness.

In the original BAN logic, the nonce verification rule is:

$$\frac{\text{P believes } \#(X),\ \text{P believes Q said X}}{\text{P believes Q believes X}}$$

From this we draw the conclusion that Q believes X, which is a nonce. However, as Mao *et al.* point out, it does not make sense to believe a nonce. One can believe that X is a nonce, or that nonce X is fresh, but one cannot believe a nonce.

Mao and Boyd claim that the method for determining assumptions in a protocol is flawed in the BAN logic. A slight modification to the assumptions could turn a useless protocol into a valuable one or vice versa. Also, there is no way to know if the assumptions are the weakest ones possible. This is obviously desirable.

The last defect of the BAN logic discussed by Mao *et al.* has to do with confidentiality. The authors use the Nessett example [50] to show that a BAN analysis fails to recognize some protocols that give away secrets to attackers. Thus, those pieces of information that must remain secrets must be explicitly designated as such.

The new logic presented by Mao and Boyd requires strict typing of formulas and messages. Thus, a principal may no longer believe a nonce, because a nonce is not of a type that may be believed. In addition, a new idealization process is defined. This process is mostly mechanical, and requires little human intervention. However, as some part of the process requires non-automated judgement, this method still suffers from the very criticism the authors have of the original BAN logic.

In the new idealization, challenges and responses are linked. The human interaction is to determine the types of the various parts of messages, and which responses to associate with the challenges. A new construct, $sup(Q)$ is defined to represent the fact that Q is a "super" principal. This means that Q is entirely trusted. An example of this would be an authentication server in the Needham and Schroeder protocol [48]. This is necessary because a principal should not be trusted only on certain beliefs. If a principal is trusted, it must be entirely trusted.

New inference rules are added in the Mao and Boyd logic. They are very similar to rules in the BAN logic, but include the *sup* construct, and are based on the mechanical idealization process. However, the reasoning process is quite different. Mao and Boyd used the *tableau* method [20]. The reasoning starts with the desired

conclusion, such as A *believes* A $\overset{K}{\leftrightarrow}$ B, and finds rules that lead to that conclusion. Thus, reasoning proceeds backwards until the initial assumptions are found. This method finds the weakest pre-conditions if necessary conditions are always found when searching for rules to apply.

The new logic is applied to the Nessett protocol to uncover the known weakness. Then, a revised version of Otway-Rees is given, and the new logic is used to prove that no conditions are violated when applying rules from the assumptions to the conclusions. Because the logic is not complete, this is the strongest statement that can be made from such a system.

The system described by Mao and Boyd reasons monotonically, as does the BAN logic, so there is no provision for refutation of belief. Human intervention still exists but is much more limited. While this logic is an improvement over previous ones, it does not provide a mechanism for proving that a protocol is correct.

3.8.3.3 Extending BAN to deal with PKCS

To analyze a public key crypto system (PKCS), the CCITT X.509 Strong Two-Way Authentication Protocol, Gaarder and Snekkenes [21] extend the BAN logic. The following constructs are introduced to represent public key cryptography and time[6]:

$\mathcal{P}K(K,U)$ The entity U has associated the good public key K.

$\Pi(U)$ The entity U has associated some good private key. The key value is only known by U.

$\sigma(X,U)$ The formula X is signed with the private key belonging to U.

$(\Theta(t_1,t_2),X)$ X holds in the interval t_1,t_2. The creator that uttered the time-stamped message X claims that X is good in the time interval between t_1 and t_2.

$\Delta(t_1,t_2)$ The local unique Real Time Clock (RTC) shows a time in the interval between t_1 and t_2.

[6]Recall that the original BAN logic has no provisions for reasoning about time.

According to the rules for public key cryptography, we have the rule

$$\frac{U_i \text{ sees } \sigma(X, U_j)}{U_i \text{ sees } X}$$

which states that any principal seeing a formula signed with another principal's private key, can see that formula.

The following rule deals with duration stamps. That is, a formula is good for a certain interval. This is necessary because the CCITT X.509 protocol being analyzed involves the use of time and a real clock.

$$\frac{\begin{array}{c}\text{P } \textit{believes } \text{Q } \textit{believes } \Delta(t_1, t_2), \\ \text{P } \textit{believes } \text{Q } \textit{said } (\Theta(t_1, t_2), X)\end{array}}{\text{P } \textit{believes } \text{Q } \textit{believes } \text{X}}$$

This is similar to one of the rules in the original BAN logic, but it restricts the time during which this rule can be applied to a "good" interval. Calvelli and Varadharajan [11] reason about time in a similar way by introducing a rule of the form $(B_i \wedge t) says\ r$ that means that B_i *says* r at or before time t.

Gaarder *et al.* also introduce the construct

$$\mathcal{R}(P, X)$$

to say that P is the intended recipient of message X. An example of this would be

$$P \rightarrow Q : \mathcal{R}(Q, X), X$$

where $\mathcal{R}(P, X)$ is a *tag* telling Q that he is the intended recipient of the message.

The authors then proceed to formalize the goals of the protocol, and idealize the protocol. The extended BAN logic is then used to prove that the protocol meets its goals.

3.8.3.4 Adding probabilistic reasoning to BAN

BAN logic is "...concerned with evaluating the trust that can be put on the goal by the legitimate communicants using beliefs of the principals," according to Campbell *et al.* [12]. It "has no provisions for modeling insecure communication channels or untrustworthy principals and, in fact, fails to model any type of insecurity."

Campbell *et al.* extend the BAN logic using probabilistic reasoning to calculate a *measure* of trust rather than complete trust. That is, given assumptions about the level of trust among principals, and a protocol, we can analyze the level of trust that this protocol achieves.

The authors define the analysis problem in terms of an equivalent linear programming problem as follows:

Let $p_1, ..., p_n$ be an assignment of probabilities to the assumptions $a_1, ..., a_n$ of a proof of the conclusion c. Then, $L \leq P(c) \leq U$, and the lower limit L (respectively upper limit U) can be obtained by solving the linear program:

minimize (resp. maximize) $z = q \cdot \pi$
subject to the constraints $W\pi = p$

$$1 \cdot \pi = 1, \pi \geq 0$$

The simplex algorithm can then be applied to find a minimal solution. This method is then applied to the Needham and Schroeder protocol. This method easily reveals the known weakness. It also shows that the assumption that K_{ab} is a good key is responsible for this weakness.

The weakness of the Needham and Schroeder protocol was discovered by the extensions of Campbell *et al.*, as the original BAN logic does. However, their method requires preprocessing of the protocol with the BAN logic, and is thus not capable of independently discovering the flaw. In addition, this application of the method of Campbell *et al.* does not constitute a proof that the protocol is secure, due to its incompleteness. The contribution of Campbell *et al.* is in obtaining the functional representation for the lower bound on the probability of conclusion in terms of the probabilities attached to the assumptions and rule instances used in the proof.

One weakness of schemes such as that presented by Campbell *et al.* [12] is difficulty of use. The original BAN logic has been praised for its simplicity, so it seems, and intuition also dictates, that there is a tradeoff between the ease of use of an analysis tool and its utility.

3.8.4 The CKT5 Logic

Bieber [4] extends the epistemic logic of Hintikka [25]. This new logic of communication in a hostile environment, called CKT5, allows a user to describe the

states of knowledge and ignorance associated with the communication via encrypted messages. Bieber also extends the logic of knowledge and time, KT5, of Sato [56] with operators that relate directly to the sending and receiving of messages.[7]

To describe a protocol, P, in CKT5, we define φ to be the way principals behave when participating in P, and ω states who knows what when the protocol terminates. Then, $\varphi \rightarrow \omega$ is a CKT5 formula that describes P. Next, it must be proved that $\varphi \rightarrow \omega$ is a theorem. An example of a member of φ is "if principal A has sent m encrypted under K, then he must have received m at some point in the past." Similarly, ω contains statements such as "At time t, A knows that k is a crypto key."

CKT5 extends the basic epistemic logic by adding modal operators to express the transmission and receipt of messages. The usual connectives are used, $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$, along with the quantifiers, $\forall, \exists$, and equality of terms ($=$). In addition, Bieber defines modal operators $K_{A,t}, R_{A,t}$ and $S_{A,t}$. If A and B are principals, t is a number representing time, and φ is a well formed formula in CKT5, and m, n, m_1, and k are terms, then the following are definitions in CKT5.

$K_{A,t}\varphi$ At time t, A knows that φ holds.

$R_{A,t}\varphi$ At time t, A received some message stating that φ holds.

$S_{A,t}\varphi$ At time t, A sent some message stating that φ holds.

$m.m_1$ The concatenation of m and m_1.

$\mathbf{d}(k, m)$ The decryption of m with key k.

$\mathbf{nonce}(n, t, A)$ n is a nonce generated at time t by the system at the request of A.

$\mathbf{private}(t, \{A, B\}, k)$ k is a symmetric key shared by A and B.

$\mathbf{key}(k)$ k is a symmetric key.

$K_{A,t}\mathbf{msg}(m)$ At time t, A knows that m is a computable term[8].

---

[7] The following summary of Bieber's logic borrows examples from Snekkenes [63], who gives an excellent summary of CKT5, and from the original paper by Bieber [4].

[8] Bieber defines a computable term as a clean message, or a concatenation of two not so clean terms (containing some encrypted part), or the result of performing a cryptographic function on a not so clean term [4].

belongs(m_1, m_2) m_1 occurs as a subterm of m_2.

clear(m) m is a clear text message.

clean(m) m is a clear text message, the concatenation of two clean messages or can be decrypted to yield a clean message.

n_s_clean(m) m contains a subterm that is not necessarily well built[9].

The logic assumes uncertain communication, so that if a message is intercepted by an intruder, it may not arrive at its destination. Messages are not lost, but the intruder can prevent them from reaching a target.

Bieber defines *univoque* messages to be ones that for agent X at time t, are well built with exclusively clear text messages and keys known only by X at time t. Formally:

$$\begin{aligned}&\text{univoque}(X,t,m) \leftrightarrow K_{X,t}\ \text{clear}(M)\\ &\quad \vee\ (\exists k\ K_{X,t} key(k) \wedge\ \text{univoque}(X,t,d(k,m)))\\ &\quad \vee\ (\exists m_1\ \exists m_2\ m = m1.m2 \wedge\ \text{univoque}(X,t,m_1) \wedge\\ &\qquad \text{univoque}(X,t,m_2)\ )\end{aligned}$$

Thus, X knows that a message is usable if it is univoque.

Bieber makes the recommendation that knowledge rather than belief be used to guarantee security because epistemic logics are better at describing the behavior of other agents, as is seen by a strong logic such as CKT5. Syverson makes a similar recommendation [66].

Snekkenes gives an example application of CKT5 [63] with KP, a protocol similar to the Needham and Schroeder protocol. CKT5 is modified so that it can distinguish between the role of a principal and its name. This is done simply by introducing a predicate "role-R" that maps principals to their roles in a protocol.

The proof that KP is secure points out the weakness of CKT5. As it is known that KP has a flaw (discussed in section 3.4.1), the fact that it can be proved correct in CKT5 demonstrates that strictly epistemic logics, as we know them, are not sufficient for analyzing the security of authentication protocols.

---

[9] Bieber defines a well built term as a clear text term, or the concatenation of two clear text terms, or the encryption or decryption of a clear text term [4].

3.8.5 Analysis of Belief Evolution

An authentication protocol analysis can be viewed as the evolution of the beliefs of the principals involved. Kailar and Gligor [31] present a logic similar to the BAN logic for reasoning about the evolution of belief within a protocol run. The types of protocols that can be analyzed with this logic include interdomain authentication, and protocols where trust in an encryption key must be established despite the lack of jurisdiction.

In this logic, beliefs in a protocol run evolve as in a state machine, where a current belief and an action determine the next state of belief.

$$\text{Belief} + \text{Action} \Rightarrow \text{New Belief}$$

The concept of a *knowledge set* for each message content is introduced. A knowledge set is a set of principals who *know* the contents of a message, a, and a given round in the protocol, M_i. Here, M_i stands for message M at instant i in the protocol run.

A message is represented as:

$$\{Message\ round, Sender, Receiver, Contents\}$$

The sender and receiver field correspond to signing with a given key, or in the case of symmetric keys, to the encryption with a session key. Thus,

$$Y \triangleright \{M_k, Y, X, C\}$$

denotes that Y sends message M with content C in round k of this session to principal X. Similarly,

$$X \triangleleft \{M_k, Y, X, C\}$$

denotes that X sees message M in round k and knows that it is sent by Y. It also reads the message contents C.

Kailar and Gligor's logic represents trust explicitly, thus avoiding some of the problems that arise in the BAN logic based on trust assumptions. The statement $TRUST_R(P, Q)$ means that P trusts principal Q in the context R. Trust means that if Q says X, and P trusts Q, then P believes X. In the analysis, a forwarded message is viewed as being sent from the originator directly to the destination. As the intermediary cannot read the contents, this short circuit makes sense.

One noteworthy assumption made in this logic is that principals can distinguish between messages from a current session, and messages from other sessions. Without this, an inconsistent state of beliefs can be attained due to unrelated message histories.

The logic contains inference rules similar to those of the BAN logic. Most of the rules are concerned with maintaining the knowledge sets, and these sets are what allow principals to reason about the evolution of other principals' beliefs in a protocol. The first inference rule presented is the belief in uniqueness of message receipt. This rule states that if X sends a message, M_i, to Y, and X believes that Z reads the message, then X believes that $Z = Y$.

$$\frac{\begin{array}{c} X \triangleright \{M_i, X, Y, C\}, \\ X \text{ believes } \{Z \triangleleft \{M_i, X, Y, C\}\} \end{array}}{X \text{ believes } \{Z = Y\}}$$

The following rule defines how knowledge sets are maintained. Here, $KS(a, M_i)$ stands for the knowledge set of contents a of message M_i, and this set contains the principals who know a after having received message M_i. Kailar and Gligor use C to denote the contents of a message, except in the case of knowledge sets where a is used. We follow their conventions.

$$\frac{\begin{array}{c} X \text{ believes } KS(a, M_i) = \{S_1\}; \\ X \text{ believes } KS(a, M_j) = \{S_2\} | j \geq i > 0 \end{array}}{\begin{array}{c} X \text{ believes } \forall Y \in \{S_2\} \Leftrightarrow \{S_1\}; \\ \exists P, k | P \in S; i < k \leq j; Y \triangleleft \{M_k, P, Y, a\} \end{array}}$$

This states that if the knowledge set for some contents a in a later round number contains principals that are not in the knowledge set of an earlier round number, then those principals must have received a message with that contents during the time interval between the rounds.

Other rules in the logic describe when a formula can be included in a knowledge set, the freshness of nonces, and the freshness of message content. They are similar in style to the ones above and can be found in the original paper [31].

Kailar and Gligor compare the use of their logic to the BAN logic for the analysis of some well known protocols such as an inter-domain authentication protocol, a PROXY ticket forwarding protocol, and a multiparty session protocol.

They show that their analysis preserves the accumulation of beliefs of all the principals in the system, whereas the BAN analysis falls a bit short.

Calvelli and Varadharajan [11] use this logic to analyze some delegation protocols for Kerberos, which is evidence of the usefulness of the logic. Others have also used it to analyze new systems. The ease of use of Kailar and Gligor's logic is seen by its applicability to many problems. This advantage is significant as is seen from the complexity and resulting lack of use of methods in the Type IV approach discussed later.

3.8.6 Semantics of Logics of Authentication

The utility of formal protocol analysis is limited by the quality of the tools we are using. Just as we have formal methods for evaluating protocols, it is useful to be able to reason about the tools themselves.

According to Syverson [67], "One of the main roles of a semantics is to give us a means to evaluate our logics. When evaluating a logic we are primarily interested in two questions: Can we derive everything we want? (Completeness) And, can we avoid deriving things we don't want (Soundness)." In general, we seem to be more concerned with soundness, whereas, for computer security, "completeness is of the utmost importance" [67].

The reason we need to ensure that we can derive anything possible is that many logics rely on the generation of all possible security flaws. If a logic is incomplete, (as is the original BAN logic [9]), then there may be flaws that are overlooked. "A formal semantics provides a precise structure with respect to which soundness and completeness of a logic may be proven" [67]. However, as Syverson explains, the semantics must not be derived directly from the logic. An independently derived semantics for a logic serves as a valuable tool in evaluating the logic.

3.8.6.1 A semantics for the BAN logic

Abadi and Tuttle [2] define a semantics for the BAN logic. They define belief as a form of resource-bounded defeasible knowledge, using a possible-worlds semantics. First, they remove some unnecessary assumptions in the original assumptions by introducing new constructs. Then, the semantics are formally defined.

The original BAN logic assumes that principals are honest, in that they believe in the truth of the messages they send. To remove this assumption, a new construct is introduced, 'X', which is read "*forwarded* X," that is used for messages that were not constructed by the principal sending them. Another construct introduced before the semantics are defined is "P *says* X" to represent the fact that P has sent X in the present. Using this, a new postulate is introduced that states that if P said X and X is fresh, then P *says* X. This promotes from knowledge to belief.

Another construct deals with shared secrets. If P and Q share a secret, s, then $\langle X^Q \rangle_s$ represents the combination (usually concatenation) of X and s. This is usually used to demonstrate knowledge of a secret.

The BAN logic is reformulated to define the semantics precisely. For the complete description, the reader is referred to the paper [2]. We give a summary of the more important aspects of the semantics. The following actions are defined for a principal P:

send(m, Q) denotes P's sending of the message m to Q. The message m is added to Q's message buffer.

receive() denotes P's receipt of a message. Some message m is nondeterministically chosen and deleted from P's message buffer.

newkey(K) denotes P's coming into possession of a new key. The key K is added to P's key set.

seen-submsgs$_K(M)$ is defined as the union of the set $\{M\}$ and

1. $\mathit{seen\text{-}submsgs}_K(X_1) \cup \cdots \cup \mathit{seen\text{-}submsgs}_K(X_k)$if $M = (X_1, \cdots, X_k)$
2. $\mathit{seen\text{-}submsgs}_K(X)$ if $M = \{X^Q\}_K$ and $K \in \mathcal{K}$[10]
3. $\mathit{seen\text{-}submsgs}_K(X)$ if $M = \langle X^Q \rangle_s$
4. $\mathit{seen\text{-}submsgs}_K(X)$ if $M =$ 'X'

said-submsgs$_{K,M}(M)$ This is defined almost the same way as $\mathit{seen\text{-}submsgs}_K(X)$ except that the fourth condition also stipulates that $X \notin \mathit{seen\text{-}submsgs}_K(X)$.

[10] P's key set.

Next, Abadi and Tuttle describe the syntactic restrictions on a protocol run, r, a time k, a key set $\mathcal{K}$, a principal P, and M, the set of messages P has received before time k.

1. A principal's key set never decreases: If $\mathcal{K}'$ is P's key set at time $k' \leq k$, then $\mathcal{K}' \subseteq \mathcal{K}$.

2. A message must be sent before it is received: If $receive(M)$ appears in p's local history at time k, then $send(M, P)$ appears in some principal Q's local history at time k.

3. A principal must possess keys it uses for encryption. Suppose that action $send(M, Q)$ appears in P's local history at time k and that $\{X^R\}_K \in said\text{-}submsgs_{K,M}(M)$. Then, either $\{X^R\}_K \in seen\text{-}submsgs_K(M)$ or $K \in \mathcal{K}$.

4. A system principal sets "from" fields correctly: if $send(M, Q)$ appears in P's local history at time k and $\{X^R\}_K \in said\text{-}submsgs_{K,M}(M)$, then $P = R$ or $\{X^R\}_K \in seen\text{-}submsgs_K(M)$. Similarly, if $send(M, Q)$ appears in P's local history at time k and $\langle X^R \rangle_Y \in said\text{-}submsgs_{K,M}(M)$, then $P = R$ or $\langle X^R \rangle_Y \in seen\text{-}submsgs_K(M)$.

5. A system principal must see messages it forwards: if $send(M, Q)$ appears in P's local history at time k and 'X' $\in said\text{-}submsgs_{K,M}(M)$, then $X \in seen\text{-}submsgs_K(M)$.

Once the syntax has been defined, the semantics can be described. The definition of $(r, k) \models \varphi$ is inductive on the structure of φ. An interpretation π maps each $p \in \Phi$ to the set of points $\pi(p)$ at which p is true. So,

$(r, k) \models p$ iff $(r, k) \in \pi(p)$ for primitive $p \in \Phi$

$(r, k) \models \varphi \cap \varphi'$ iff $(r, k) \models \varphi$ and $(r, k) \models \varphi'$

$(r, k) \models \neg\varphi$ iff $(r, k) \not\models \varphi$

Next, the semantics are described for the various constructs in the logic. For example $P\ sees\ X$ at (r, k) is defined as

$$(r, k) \models P\ sees\ X$$

iff, for some message M, at time k in r

- $receive(M)$ appears in P's local history
- $X \in seen\text{-}submsgs_K(M)$, where K is P's key set.

Also, P has jurisdiction over φ at (r, k) is defined as

$$(r, k) \models P \text{ controls } \varphi$$

iff $(r, k') \models P \; says \; \varphi$ implies $(r, k') \models \varphi$ for all $k' \geq 0$.

The other constructs in the BAN logic are defined similarly in the semantics. The notion of belief is captured using a possible-worlds semantics where a principal believes a fact if that fact is believed in all the possible worlds known to that principal at that time. Abadi and Tuttle [2] prove that this axiomization is sound, but state that they doubt it is complete. They give an example of a formula that is valid, but cannot be generated using the logic:

$$(P \; controls \; (P \text{ has } K) \; \wedge$$
$$P \text{ says } (P \text{ has } K, \{X^P\}_K)) \supset P \text{ says } X$$

In the semantics described by Abadi and Tuttle, choosing good protocol runs is important. They do not allow an initial assumption with a negative belief, such as P_i does not believe K is a good key. This seems to be a reasonable assumption. As the authors state, "In every application of this logic that we are aware of, the initial assumptions satisfy this restriction."

3.8.6.2 A semantic model for authentication protocols

Woo and Lam [80] present a semantic model for authentication protocols. They identify *correspondence* and *secrecy* as two correctness properties. Correspondence specifies that different principals in a protocol must execute steps in a locked-step fashion. This represents the idea that a protocol step can be in response to a previous protocol step, and not just an independent event.

The authors define an action schema to specify the steps in a protocol. In protocol specification, each of these actions is preceded by a label. The actions allowed are:

BeginInit (r) NewNonce (n)

EndInit (r)	NewSecret (S, n)
BeginRespond (i)	Send (p, M)
EndRespond (i)	Receive (p, M)
Accept (N)	GetSecret (n)

The meanings of these actions are for the most part intuitive. A notable exception is the *GetSecret* action. This is used to model the compromise of an old key by the intruder. The action would not be included in a protocol specification, but rather, on the consequence side of a rule, and serves to eliminate timeliness requirements.

A protocol specification begins with a set of initial conditions, followed by the protocol for each participant. For example, the authors specify the Otway-Rees protocol [51] using their model. The specification takes the form.

1. Initial Conditions
2. Initiator Protocol
3. Responder Protocol
4. Server Protocol

Notice that although the Otway-Rees protocol does not differentiate between the communicating principals, Woo and Lam explicitly designate the roles as initiator and responder.

The main difference between this work, and that of Syverson [66] is that Woo and Lam specify protocols as programs and are concerned with a general formalism of correctness, whereas Syverson is more concerned with logic. The approach by Woo *et al.* is revolutionary in that it recognizes and formalizes the notion of correspondence in authentication protocols.

3.8.7 A Nonmonotonic Logic of Belief

All of the logics we have discussed so far have dealt with monotonic knowledge and belief. However, in a real world model, our beliefs can change. For example, if a session key is compromised, we need to change our belief that this is a good key.

Moser [46] describes a nonmonotonic logic of belief based on a monotonic logic of belief and knowledge. She describes the standard S5 [13] knowledge axioms. Here $K_i(p)$ means that principal i *knows* p.

$B_i(p)$	$B_i(q)$	$B_i(p)$ unless $B_i(q)$
t	t	f
f	t	t
t	f	t
f	f	x

Figure 3.5: **The Definition of Moser's *unless* operator** The x in the last row indicates a special case. x is true iff $\exists r : B_i(p)$ unless $B_i(r) \in F$, where F is a conjunction of formulas containing the *unless* operator.

1. $K_i(p) \Rightarrow p$
2. $K_i(p) \wedge K_i(p \Rightarrow q) \Rightarrow K_i(q)$
3. $\neg K_i(p) \Rightarrow K_i(\neg K_i(p))$ (Negative introspection)
4. $\vdash p$ infer $K_i(p)$

Axiom 4 corresponds to the axiom **T** described by Syverson [71] (see section 3.8.1). Also, Moser points out that positive introspection is easily derivable from the above axioms. The axioms for belief are the standard KD45 axioms [13], and are the same as those described by Rangan [52] (see section 3.8.1).

In Moser's logic, a belief is considered true unless it is stated otherwise. She introduces a new predicate, *unless* whose value can be seen from the following truth table (where F is a conjunction of formulas containing the *unless* operator).

The definition of *unless* is given by the truth table in Figure 3.5. The x in the last row is the most interesting part of the definition and indicates a special case. x is defined as follows:

$$x = \begin{cases} t & \text{if } \exists r : B_i(p) \text{ unless } B_i(r) \in F \\ & \text{and } B_i(r) \text{ is true} \\ f & \text{otherwise} \end{cases}$$

Thus, the value of the *unless* operator depends on the context in which it appears. This definition allows for any belief to be held unless it is refuted somewhere else in the formula.

Moser proceeds to give an application of this logic for a key distribution protocol. Although her logic provides for a new type of reasoning, there are a few shortcomings. Moser does not discuss the tractability of her logic other than pointing out that if quantification were added to the logic, it would be undecidable. Also, Moser makes no mention of soundness and completeness. Perhaps a formal semantics for this logic would help answer such questions.

Another shortcoming of Moser's logic is that it deals with the nonmonotonicity of belief, and mentions nothing of the nonmonotonicity of knowledge. In fact, there is no way to reason about a principal forgetting some information. An example of such a protocol is the *khat* system of Chapter 2. The security of *khat* is based on the notion that a vacant workstation erases some information from its memory so that an intruder gains nothing from compromising the machine. To reason about such systems, we need a nonmonotonic logic of knowledge as well as belief. Moser's logic does not provide this.

3.9 Type IV Approach

The Type IV approach to protocol analysis develops a formal model based on the algebraic term-rewriting properties of cryptographic systems. This approach was introduced by Dolev and Yao [19], and has since been pursued by Merritt [43], Syverson [65, 70], Meadows [39, 40, 41, 42], and Woo & Lam [80]. The more recent applications of this approach have provided automated support for the analysis, and have enabled a user to query the system for known attacks.

The Type IV approach generally involves an analysis of the attainability of certain system states. In this regard, it is similar to some of the Type II approaches discussed earlier. However, the Type IV approaches try to show that an insecure state cannot be reached, whereas the Type II approaches began with an insecure state and attempted to show that no path to that state could have originated at an initial state.

3.9.1 Dolev and Yao

Dolev and Yao [19] proposed the first algebraic model for the security of protocols. Their protocols dealt more with the distribution of secrets than authentication, although the two are closely linked. The main difference is that we generally

think of authentication as involving a third party authentication server, whereas the Dolev and Yao protocols dealt with only two parties.

Dolev and Yao define some classes of protocols. They reason about these classes of protocols rather than individual protocols themselves, and prove some interesting properties of these classes. For example, *cascade protocols* and *name-stamp protocols* are examined. A cascade protocol is one in which a user can apply the public key encryption-decryption operations in several layers, to form messages. The authors prove that such protocols are secure if and only if the following conditions hold:

1. The messages transmitted between X and Y always contains some layers of encryption functions E_x or E_y

2. In generating a reply message, each participant A ($A = X, Y$) never applies D_A without also applying E_A.

Similarly, Dolev and Yao provide a polynomial-time algorithm for deciding if a given name-stamp protocol is secure.

Dolev and Yao not only show how to model protocols algebraically, they consider whole classes of protocols and demonstrate how to reason about any protocol that shares certain properties.

3.9.2 Merritt's Model

Merritt [43] generalizes on the technique of Dolev and Yao to model diverse cryptographic systems and to formally state and prove security properties other than secrecy. His approach is to use the the messages an attacker knows as well as the relationship between these messages to model its knowledge.

Merritt defines a `partial algebra`, $A = \langle D, R_1, \cdots, R_k \rangle$, where D is a set, and $R_1, \cdots, R_k$ are relations on D. A `subalgebra` of A is any algebra A' with $D' \subseteq D$, and whose relations are the restrictions to D' of the relation in A. Algebras such as $B = \langle D, R_1 \rangle$ containing a subset of the relations in A are called `reducts` or A.

Let $I = \langle M, C, E_A, E_B, e_A, e_B, {e_A}^{-1}, {e_B}^{-1} \rangle$ be a partial algebra where M is a set of messages, $C \subseteq M$ is the set of cleartext messages, E_X is a predicate of the

principal X that is true only on messages which are properly encrypted, e_X is the encryption function of principal X, and ${e_A}^{-1}$ is the decryption function of principal X. The reducts $I_A = \langle M, C, E_A, e_A, {e_A}^{-1}, e_B \rangle$ and $I_B = \langle M, C, E_B, e_B, {e_B}^{-1}, e_A \rangle$ represent the cryptographic capabilities of A and B.

Merrit shows how this algebraic model can be used to analyze cryptographic protocols by performing algebraic operations using algebras such as I above. In the next section we see how an algorithm can be used to further analyze the behavior of the intruder.

3.9.3 Using the NARROWER Algorithm for Protocol Analysis

According to Meadows [39], "A cryptographic protocol may be thought of in part as a set of rules for generating words in some formal language." We can define algebraic operations on these words, such as encryption and decryption. The security of a protocol can then be based on the ability of an intruder to generate certain words in the language.

The operations in a term-rewriting system are the reductions of terms using the cancellation properties of the words in the system. Examples of such rules are:

1. $d(X, e(X, Y)) \rightarrow Y$
2. $e(X, d(X, Y)) \rightarrow Y$

which define the symmetric properties of encryption and decryption. An intruder can attempt to see if any words available to him can reduce to some secret word, say a session key.

Meadows [39] uses the NARROWER [53] algorithm, which she has implemented in Prolog [15] to analyze the IBM key management scheme [16] mentioned in Section 3.7.2. The algorithm begins with a trivial set (possibly the empty set) of words available to the intruder, and an initial state. A set of secrets, which the intruder should not learn, is also defined. The algorithm attempts to show that there is no path through the protocol, beginning at the initial state, that leads to a state where the intruder can learn words in the secret set.

The algorithm works by induction on the length of the path, beginning with the trivial set, and continues until no more paths can be generated. For any m, we can state that no "dangerous" path of length less than or equal to m exists. The user can interact with the program to improve on the tractability of the problem by modifying the initial set, and the rules available.

In a later paper [41], Meadows analyzes the Burns-Mitchell resource sharing protocol [8] using an analysis tool based on the same term-rewriting properties utilized by the NARROWER. This system models the knowledge and belief of the intruder, and defines a set of rules whereby an intruder can learn new information based on protocol steps. Using this technique, Meadows demonstrates the existence of a flaw in the Burns-Mitchell protocol. Meadows suggests a fix to this protocol, and then uses the analysis technique to show that the attack no longer succeeds.

In the analysis of the IBM key management scheme, it is shown that certain secrets are unobtainable by a penetrator unless a session key has been compromised. However, such a proof is not a proof that the protocol is *secure*; this would require proving that the term-rewriting system is complete, *i.e.*, that every valid word can be generated. Unfortunately, the system is not complete. Another requirement for proving that a protocol is secure is that the method for formalizing the protocol, must itself be formal, and it is not.

Thus, methods such as using the NARROWER can be used to find insecurities in a protocol, but do not constitute a proof that a protocol is secure.

3.9.4 The KPL Logic

In the logics presented so far, we have seen ways of representing the fact that P knows that K_{PQ} is the secret key between P and Q. However, there has been no way to represent simply the fact that intruder Z knows P's key. Syverson calls this a key *simpliciter*, and his KPL logic [65, 66] can represent such a fact.

KPL is a quantified modal logic with a corresponding possible-worlds interpretation. In KPL, Z knows P's key if P's key is present in all of the worlds accessible to Z from his current set of possible worlds via some transition. Syverson defines a semantics for the logic that he uses to prove the soundness and completeness of KPL.

As Syverson states, the soundness and completeness of KPL do not guarantee that there can be no error in the reasoning about a secure protocol, but they do prove that there can be no formal error: "Once we have formally specified a protocol, a logical derivation of any result concerning the specification will be correct — *i.e.* true of that specification — and anything that can be formally shown to be a semantic consequence of that specification will be provable in the logic."

Syverson does not provide examples of how to specify an authentication protocol in KPL; such a specification would be complicated. In general, the Type IV approaches suffer from a great deal of complexity, and thus their value as an analysis tool is diminished.

3.9.5 The NRL Protocol Analyzer

Syverson and Meadows [70] use the techniques described above, namely, using the term-rewriting properties of protocol specifications, to develop the NRL protocol analyzer. This system is used to analyze classes of protocols, and is not tied to any assumptions about the protocols.

The NRL protocol analyzer allows a single set of requirements to specify a class of protocols. The following symbols are used:

$\rightarrow$ represents the standard conditional

$\wedge$ represents conjunction

$\diamond$- represents a temporal operator meaning *at some point in the past.*

Each principal keeps track of a local round number for a protocol, and the following actions are defined:

accept(B, A, Mes, N) means that B accepts the message Mes as from A during B's local round N. N is an optional parameter.

learn(Z, Mes) means that the intruder, Z, learns the word Mes.

send$(A, B, (Query, Mes))$ means that A sends Mes to B in response to $Query$. The use of a $Query$ is optional.

request$(B, A, Query, N)$ means that B sends $Query$ to A during B's local round N. $Query$ is optional.

From these constructs and actions, requirements can be specified. The requirements are represented by a conjunction of statements. For example:

Requirement #1

- $\neg(\diamond \text{ accept}(B, A, Mes, N) \wedge \diamond \text{ learn}(Z, Mes))$
- accept $(B, A, Mes, N) \rightarrow$
  $\diamond$ (send$(A, B, (Query, Mes)) \wedge$
  $\diamond$ request$(B, A, Query, N))$

Requirement #1 contains two conditions, both of which must hold. The first condition is that if B accepted message $Mess$ from A at some point in the past, then the intruder did not learn $Mess$ at some point in the past. The second condition is that if B accepted message $Mess$ from A in B's local round, N, then A sent $Mess$ to B as a response to a query at B's local round N.

It is clear from the last sentence why formal methods are needed to represent such statements. There is a need for a precise description. If it is not necessary for A to send the message in response to B's query *only* after the query, then we can have the relaxed requirement:

Requirement #2

- $\neg(\diamond \text{ accept}(B, A, Mes) \wedge \diamond \text{ learn}(Z, Mes))$
- accept $(B, A, Mes, N) \rightarrow$
  $\diamond$ (send$(A, B, Mes)) \wedge \diamond$ request$(B, A, N))$

The omission of the $Query$ from the *send* and *request* actions are due to the relaxation of the requirement. Also, it may be required that the messages from A and B be recent. These can be specified with the following requirement:

Requirement #3

- $\neg(\diamond \text{ accept}(B, A, Mes) \wedge \diamond \text{ learn}(Z, Mes))$
- accept $(B, A, Mes, N) \rightarrow$
  $\diamond$ (send$(A, B, Mes)) \wedge \diamond$ request$(B, A, N))$

- accept $(B, A, Mes, N) \rightarrow$
  $\diamond$ (send$(A, B, Mes)) \wedge$
  $\neg(\diamond$ time_out$(B, N)) \wedge$
  $\neg(\diamond$ time_out$(A, N))$

The new action, *time_out*, is used to control currency of messages.

To avoid assumptions such as those made in the BAN logic [9] that all participants in a protocol are honest, an honest user A is designated as $user(A, honest)$, and a dishonest user as $user(A, dishonest)$, and in the case where it is not known, a variable Y can be included, $user(A, Y)$. To specify the requirement that an honest B accept a message as coming from an honest A only if it was never previously accepted by an honest user, Syverson and Meadows [70] provide the following complicated requirement:

Requirement #4

- $\neg\diamond$ accept$(user(B, honest), user(A, honest),$
  $Mes) \vee\neg\diamond$ learn(Z, Mes)
- accept$(user(B, honest), user(A, honest), Mes, N)$
  $\rightarrow \diamond$ (send $(user(A, honest), user(B, honest), Mes)$
  $\wedge\diamond$ request $(user(B, honest), user(A, honest), N))$
- accept$(user(B, honest), user(A, honest), Mes) \rightarrow$
  $\neg\diamond$ accept$(user(C, honest), user(D, Y), Mes)$

The variable Y states that the message could not have been accepted by anyone, regardless of honesty.

The NRL protocol consists of a state space, where a protocol trace is an infinite sequence of states. A *model* is an ordered 4-tuple, $\langle S, I, \sigma, t\rangle$ such that S is a state space, I is an interpretation, σ is a trace, and t is time. Satisfaction is defined as $\langle S, I, \sigma, t\rangle \models \alpha$ means that α is true at $\langle S, I, \sigma, t\rangle$. Syverson and Meadows [70] give a detailed definition of the $\models$ relation in their paper.

A specification in the NRL protocol analyzer is modeled on the communication of honest participants. Dishonest participants are assumed to be modeled by the intruder, and so are not modeled separately. Each honest participant possesses a set of learned facts, called *lfacts*. Also, a function called *intruderknows()* represents the lfacts known by the intruder.

To use the protocol analyzer, a protocol is specified using the above constructs. Then, a set, K, of lfacts is defined. K may contain a possible attack by an intruder. The analyzer then can determine if the set K can meet the requirements of the protocol. If so, a successful attack by the intruder can be discovered.

The NRL protocol analyzer consists of four phases. In the first phase, transition rules are defined for the actions of honest principals. In phase 2, the operations available to all principals, such as encryption and decryption, are described. Phase 3 consists of describing the atoms used as the basic building blocks of the words in the protocol, and the final phase describes the rewrite (reduction) rules obeyed by the operators. An example of such a rule is:

IF:
  count(user(B,honest)) = [M],
  lfact(user(B,honest),N,recwho,[11]M) =
    [user(A, Y)],
  not(user(A, Y) = user(B,honest)),
THEN
  count(user(B,honest)) = [$s(M)$],
  intruderlearns([user(B,honest),
    rand(user(B,honest),M)]),
  lfact(user(B,honest),N,recsendsnonce,
    $s(M)$) = [rand(user(B,honest),M)],
EVENT
  event(user(B,honest),N,requestedmessage,
    $s(M)$) = [user(A,Y),rand(user(B,honest),M)].

This rule describes the sending of a nonce from an honest principal, B, to some principal A, whose honesty is unknown.

Operations are described by listing the restrictions on them (for example, key length), and then defining their properties. Similarly, rewrite rules describe the cancellation of operations. For public key encryption, for example, a rewrite rule would be:

$$\text{pke}(\text{privkey}(X),\text{pke}(\text{pubkey}(X), Y)) \Rightarrow Y$$

[11]The word *recwho* is used by the authors to mean "the principal who receives this."

Syverson *et al.* [70] then give a full specification and show that the protocol meets the specification.

The purpose of the NRL protocol analyzer is to show that a given protocol specification meets its requirements. However, this does not constitute a proof that the protocol is secure. The NRL protocol analyzer can be viewed as a tool that, combined with other tools, could eventually lead to a protocol that can be proven secure.

3.10 Conclusions

This chapter surveys the current state of research into the formal analysis of authentication protocols. The field has made substantial progress in the detection of flaws in published protocols as well as in the development of formal specification techniques. We have seen how various techniques from other fields can be used to reason about security in key management schemes. We have also seen the weaknesses of such methods in that they fail to capture the subtle properties of these protocols, such as their susceptibility to replay attacks.

Some authors have developed expert systems to experiment with various scenarios in an authentication protocol. Such systems are useful as tools in the development of protocols, but have not been able to offer much in the way of formal analysis of existing protocols. In particular, such systems are good at recognizing known attacks on protocols when they are specified, but have not been able to produce new, previously unknown attacks.

The predominant technique for analyzing cryptographic protocols is to use logical reasoning about belief and knowledge in the system. These schemes have been successful in proving that a protocol meets its formal requirements. However, a criticism of these systems is that the process of formalizing the requirements is itself not formal. To cope with this, semantics have been presented for reasoning about the logics themselves. There is a debate as to whether epistemic logics (knowledge) are preferable to doxastic logics (belief), but either one can be used to reason about the other.

Another criticism of logics based on belief and knowledge is that they are used to model trust and not security. Although the two are related, it is clear that

they are not interchangeable.

In addition to the above methods, some have used the algebraic term-rewriting properties of protocols to reason about security. This technique has been successful in uncovering flaws in known protocols. Unlike the modeling with belief and knowledge, the term-rewriting algebras are highly complex. It is doubtful that many protocol developers will be able to use these systems. In contrast, it is common to find analyses of protocols using BAN. However, the ease of use of techniques such as BAN creates a danger of misuse by people who do not fully understand their purpose and limitations, as has been frequently demonstrated.

Although we have presented numerous ways to reason about the security of protocols, and in some cases, to prove that they meet their requirements, there is no technique known for proving that a protocol is secure. The reason for this may be that security itself is not sufficiently well defined. We can prove that a protocol is correct, or that it meets its specification. We can even prove that under various assumptions, certain attacks against a protocol will not work. However, we have no general-purpose method of proving that an arbitrary authentication protocol is secure.

Future research is likely to focus on formal methods for formalizing authentication protocols. The weakest link in current proofs of security is the formalization process. We believe that once all of the aspects of a protocol can be converted to a formal specification using a sound and complete formal method, that we will then be able to assure a proven level of security.

First Author	Protocol Specification	Protocol Analysis			
		Type I	Type II	Type III	Type IV
Abadi				[2]	
Bieber				[4]	
Blumer	[5]		[5]		
Britton		[6]			
Burrows				[9] [10]	
Calvelli				[11]	
Campbell				[12]	
Dolev					[19]
Gaarder				[21]	
Gong				[22] [23]	
Gray				[29]	
Kailar				[31]	
Kasami					[32]
Kemmerer	[33]	[33]	[33]		
Longley			[36]		
Lu					[37]
Mao				[38]	
Meadows	[41]		[41]		[39] [40] [41] [42]
Merritt					[43]
Millen			[44]		
Moser				[46]	
Nessett				[50]	
Rangan				[52]	
Sidhu			[60]		
Snekkenes				[61] [62] [63]	
Syverson	[70]		[70]	[66] [67] [68] [69] [71]	[65] [70]
Varadharajan	[75] [76] [77]	[75] [76] [77]			
Woo	[80]				[80]

Table 3.1: The Focus of Research in the Specification and Analysis of Authentication Protocols by Category. Entries in the table correspond to bibliography reference numbers. The four types under protocol analysis are as described by Meadows [42].

CHAPTER 4

NONMONOTONIC CRYPTOGRAPHIC PROTOCOLS

4.1 Overview

This chapter presents a new method for specifying and analyzing cryptographic protocols. Our method offers several advantages over previous approaches.

Our technique is the first to allow reasoning about nonmonotonic protocols. These protocols are needed for systems that rely on the deletion of information. There is no idealization step in specifying protocols; we specify at a level that is close to the actual implementation. This avoids errors that might otherwise render a specification that passes the analysis, useless in practice.

In our method, knowledge and belief sets for each principal are modified via actions and inference rules. Every message is considered to be broadcast, and we introduce the `update` function to maintain global knowledge. We show how our method uncovers the known flaw in the Needham and Schroeder protocol [48], and that the revision by the same authors [49] does not contain this flaw. We also show that our method correctly handles protocols that are trivially insecure, such as Nessett's noted example. [50]

We then apply our method to our *khat* protocol (see Chapter 2). The analysis reveals a serious, previously undiscovered flaw in our nonmonotonic protocol for long-running jobs; one that seems obvious in hindsight, but escaped the attention of the authors and over 300 USENIX conference attendees. In addition, our analysis reveals a previously unknown vulnerability in phase II of *khat*. These are stunning confirmations of the importance of tools for analyzing cryptographic protocols.

4.2 Introduction

In computer networks, communicating parties must share a set of rules describing the messages they will send and receive. These rules, or protocols, are the foundation on which modern networks are built. As protocols are necessary to establish any useful communication, standard sets of rules are published and made widely available. This allows users all over the world to communicate with each other and share information on networks such as the Internet.

Unfortunately, the availability and widespread knowledge of communication protocols has also facilitated the malicious interference of active intruders on the network. To combat this, cryptographic protocols that rely on the encryption of data were developed. It is widely accepted that the security of data in networks should rely on the underlying cryptographic technology, and that the protocols should be open and available. [78] However, many protocols have been found to be vulnerable to attacks that do not require breaking the encryption, but instead manipulate the messages in the protocol to gain some advantage. The advantages potentially gained by an attacker range from the compromise of confidentiality to the ability to impersonate another user.

Analysis techniques have been developed to help discover flaws in protocols before they are trusted. Flaws were discovered in such well known protocols as the Needham and Schroeders key distribution protocol [17] and the CCITT X.509 protocol [9]. The BAN logic of Burrows, Abadi, and Needham [9] and its descendants [11, 12, 21, 23, 38, 62], have been pivotal in the ability to use knowledge and belief in the analysis of cryptographic protocols to discover flaws.

All of the logics developed to date reason monotonically. That is, once something is known, it is always known. This has been a fundamental obstacle in providing a complete logic because negation is missing. This means that there are valid formulas that cannot be derived. There have been two attempts to remedy this. Abadi and Tuttle [2] provide a semantics for the BAN logic that includes a new construct for negation. Moser [46] provides a nonmonotonic logic of belief. However, neither of these deal with nonmonotonicity of *knowledge*. The difference between nonmonotonicity of knowledge and nonmonotonicity of belief is discussed in Section 4.4.

1. $A \rightarrow S : A, B, N_a$
2. $S \rightarrow A : \{N_a, B, K_{ab}, \{K_{ab}, A\}_{K_{bs}}\}_{K_{as}}$
3. $A \rightarrow B : \{K_{ab}, A\}_{K_{bs}}$
4. $B \rightarrow A : \{N_b\}_{K_{ab}}$
5. $A \rightarrow B : \{N_b \Leftrightarrow 1\}_{K_{ab}}$

Figure 4.1: **The Needham and Schroeder protocol specification.** Protocols are specified by a principal name, followed by an arrow and another principal name, followed by a message.

We present a new method for analyzing protocols. This is the first method proposed for reasoning nonmonotonically about knowledge in cryptographic protocols. Our approach is a variation on the protocol specification techniques of Woo and Lam [80] where each principal's actions are defined separately. In addition, we do not require protocol *idealization*, and thus avoid many of its associated pitfalls as described by Mao and Boyd. [38] The notation we use is based on the original BAN logic [9], and we use a similar reasoning mechanism.

We show how our new method can be used to specify and analyze the Needham and Schroeder protocol. We then use it to analyze the *khat* protocol described in chapter 2, which uses nonmonotonicity of knowledge, and we show that no other analysis techniques can be used to analyze this protocol. Finally, our method is used to uncover the flaw in a famous protocol presented by Nessett [50] that he used to demonstrate a weakness in BAN logic.

4.3 Protocol Specification in a Distributed System

A typical protocol specification consists of a list of messages between principals. For example, the Needham and Schroeder protocol specification [48] can be seen in Figure 4.1. $A \rightarrow B : M$ means that principal A sends message M to principal B. A protocol designer thus specifies a protocol by listing the messages principals send to each other.

Although such a specification is intuitive, it does not represent the way

a protocol is implemented in a distributed system. In a distributed network, each principal need be aware only of his potential role in a protocol. For example, in the Needham and Schroeder protocol presented above, principal S is not concerned with messages 3, 4 and 5. In addition, there are calculations and actions (such as decryption and encryption) performed by principals during the run of a protocol that are not captured by the specification.

The need to idealize protocols before analyzing them is a weakness in this form of protocol specification. In this chapter, we propose a new method for specifying protocols that conform to the distributed system model, one that does not require idealization.

The model of Woo and Lam [80] assigns roles to the principals in a protocol and treats them as independent processes. The actions of the principals are described with no regard to the actions of others in the system. Our method for protocol specification and analysis is based on this notion.

Specifying protocols as in Figure 4.1 has another disadvantage. Protocol analysis is seen as a process separate from the specification. Current analysis techniques take a completed specification as input and attempt to reason about the completed protocol.[1] For example, Figure 3.4 is a depiction of the BAN logic. [9]

We suggest that protocol analysis should be integrated with the specification process. Thus, as a protocol is developed, beliefs and states of knowledge that represent the current state of the system are updated. At any point, an inconsistency can be detected. This has the advantage of identifying potential causes of problems as well as the actual flaws. When a protocol has been completely specified, the analysis is complete as well.

4.4 Nonmonotonicity of Knowledge vs. Nonmonotonicity of Belief

With few exceptions, previous work in the application of the logic of knowledge and belief to the analysis of cryptographic protocols has considered only mono-

[1] A notable exception is the NRL Protocol Analyzer by Syverson and Meadows [70]. However, this system is not modeled after the BAN type of reasoning about belief and knowledge, but uses the term-rewriting algebraic properties of a protocol.

tonic reasoning systems. In these systems, once something is believed, it is always believed. The same applies for knowledge.

The difference between knowledge and belief is subtle. A principal *knows* that his key is K. A principal *believes* that a nonce is fresh. In general, a principal knows things like secrets and data; a principal believes meta-data, or information about the data, such as freshness.

Monotonic systems have trouble reasoning with incomplete information. A belief that is assumed in the absence of other information, can be nullified by the introduction of new information. However, a monotonic system has no mechanism to do this. In fact, most previous systems have no refutation. The ability to refute beliefs is important for reasoning about protocols. For example, if a session key is compromised, we need to change our belief that this is a good key.

Moser [46] gives a nonmonotonic logic of belief. This logic is biased towards belief in the absence of information. Thus, a final interpretation of a formula is believed unless there is some information that makes it inconsistent. The logic uses a construct called *unless* to achieve this. This is discussed in detail in Chapter 3. We give a brief review for the purposes of this chapter.

The value of a formula using *unless* can be seen from the truth table in Figure 3.5 (where F is a conjunction of formulas containing the *unless* operator and $B_i(p)$ means that principal i believes p). The x in the last row is a special case and is defined as follows:

$$x = \begin{cases} t & \text{if } \exists r : B_i(p) \text{ unless } B_i(r) \in F \\ & \text{and } B_i(r) \text{ is true} \\ f & \text{otherwise} \end{cases}$$

The logic of Moser suffers from intractability. In addition, the logic deals only with nonmononicity of *belief*. There is no known reasoning system that deals with the nonmonotonicity of *knowledge*. A situation where such reasoning applies is a protocol that requires a principal to no longer possess information it previously knew. This is different from a principal not believing a statement it previously believed.

The *khat* protocol, described in Chapter 2, is an example of a protocol that requires reasoning with nonmonotonicity of knowledge. The protocol relies on a public workstation "forgetting" some information. The BAN logic, along with its extensions, does not provide a way for representing this behavior.

In this chapter, we introduce a method for analyzing protocols such as *khat*, where information is erased and no longer known by a principal. Our method uses *observers* sets for each secret that contain the principals who know it. These are similar to the knowledge sets of Kailar *et al.* [31]. However, Kailar *et al.* use these sets to reason about belief, whereas we apply the concept in a slightly different way to reason about the nonmonotonicity of knowledge.

4.5 The KHAT protocol

The *khat* system presented in Chapter 2 was built to solve the problem of long-running jobs in an authenticated environment where a trusted server issues tickets with limited lifetimes for services. *Khat* stands for Kerberized *at*, and is based on the UNIX *at* command. When using this service, a user schedules a job for a future time and date, with the option of renewing tickets until the job completes.

We now review the features of *khat* that are relevant to this chapter. When a user submits a *khat* job, the program creates a spool file containing everything necessary to run the job at a later date, such as environment variables, and sends it to the *khat* server. The server stores the spool file for the job, and the user's workstation erases it from memory. The *khat* client generates a new key, N, which is used to encrypt the secret key, K, that will serve as the session key when it is time for the job to run, and the server and client need to communicate. N is also stored by the server and erased by the client. The process of securing the session key, K on the client is depicted in Figure 2.7.

The *khat* protocol is initiated by the usual ticket granting method in Kerberos. A ticket for the *khat* service is granted to the client after the initial authentication. This process is well known and is believed to be secure. Toussaint provides a proof that the Kerberos ticket granting protocol is secure. [73] We take the results of the ticket granting process as the initial assumptions in our analysis. Thus, the *khat* protocol begins after ticket granting completes.

The *khat* system can be divided into two phases. Phase I works as follows.

1. A Kerberos ticket for *khat* is granted to the client and a session key is established.

2. The client generates a spool file for the job.

3. The spool file is sent to the server under the session key.

4. The server stores the spool file.

5. The client generates a new key, N, sends $\{N\}_K$ to the server, and erases the spool file and N from its memory.

6. The server stores N.

Phase II occurs when it is time for the job to run. The server wakes up once a minute to see if any *khat* jobs are ready. If so, phase II is initiated as follows:

1. The server sends N to the client.

2. The server sends the spool file to the client under the session key, K.

3. The client runs the job.

For a more complete discussion of *khat* the reader is referred to chapter 2.

It is clear that this protocol cannot be specified as a simple list of messages such as in Figure 4.1. The specification method we present in this chapter is more appropriate because we can include steps such as step 5 in Phase I.

The analysis depends on the assumptions in our threat model. We are concerned with an active intruder who has access to all network resources and can intercept, replace or delete any message. In addition, we are concerned with vacant workstations and information on them that can be useful to an intruder. This threat is also discussed in Chapter 2. In that chapter, an informal discussion of the security risks of *khat* is given. The method we present here grew out of an attempt to analyze *khat* more formally, and to provide a method for analysis of any system that must reason about nonmonotonicity of knowledge.

4.6 Specifying a Protocol

In this section we provide definitions and notation for specifying a protocol. There are two types of definitions. Those in the first type are global to the protocol, and definitions of the second type are local to each principal.

To accommodate different levels of trust among principals, we place the beliefs of the principals in the local sets. If the assumption were to be made that

each principal in the system is either trusted by everyone or trusted by no one, as is the case in many simple authentication systems, then we could have put these beliefs in the global set. To maintain generality, the level of trust and belief will be local. Thus, jurisdiction, the ability to assign session keys, is a belief that must be held by the parties sharing the keys, but not by everyone else.

The protocol designer may wish to specify and analyze a protocol for a system with untrustworthy principals. We include a *trust matrix* in the specification where the trust between each pair of principals is established. This is explained in Section 4.6.3.

As pointed out by Mao and Boyd, some statements in the BAN logic are not intuitive, such as the notion of believing a key or a nonce [38]. To remedy this, we define two local sets. One set is composed of the items that a principal possesses, such as encryption keys and nonces. The other set contains the principal's beliefs, such as the freshness of a key, or the possessions of another principal. Items in the possession sets are labeled by their origin. Each possession is accompanied by information that either states that it was generated by the principal himself, or states from whom it was received.

We define actions for dealing with the knowledge in a protocol, and inference rules for reasoning about belief. The actions are specified by the protocol designer and can be chosen from a specific set of actions defined below. Inference rules can be added by the designer although they will usually be the same across protocols.

4.6.1 Global Sets

The first step of the specification of any protocol using our method is to instantiate several sets that are needed for information that concerns the entire protocol. The are called the *global sets*. It should be noted the contents of these sets change as a protocol run is simulated in the analysis. The specification of a protocol is simply the starting point of the analysis. In this section we give the definitions of the global sets used for protocol specification. We introduce $\mathcal{W}$, for world, to represent all the principals. Also, for each set, the subscript n represents its cardinality, but this value changes from set to set.

Principal Set: This set contains the principals who participate in a protocol. $P = \{P_1, P_2, \cdots, P_n\}$. Any P_i may be marked as an initiator of the protocol. We will assume there is only one initiator.

Rule Set: This set contains inference rules for deriving new statements from existing assertions. These are the same as the inference rules in the BAN logic. $R = \{R_1, R_2, \cdots, R_n\}$ where R_i is of the form $\frac{C_1, C_2, \cdots, C_n}{D}$, C_i is a condition and D is a statement.

Secret Set: This set contains all of the secrets that exist at any given time in the system. The cardinality of this set changes during the analysis as new secrets, such as session keys, are added. $S = \{S_1, S_2, \cdots, S_n\}$.

Observers Sets: For each S_i, $Observers(S_i)$ contains all the principals who could possibly know the secret S_i by listening to network traffic or generating it themselves. The members of the *Observers* sets can be stated explicitly or maintained as formulas representing their membership.

The set, P, contains names of the participants in a protocol. A typical example might be, $P = \{A, B, AS\}$, where A and B are regular principals and AS is the authentication server.

An example of the set S is $\{K_{ab}, K_{as}, K_{bs}\}$. This set contains secret keys held among A and the authentication server, among B and the authentication server, and a session key among A and B. The session key would not be a member of S in a specification where K_{ab} is distributed in the protocol, but would be added to the set during the analysis at the point in which it was generated by the authentication server. This process is discussed in the analysis section. In this example, $Observers(K_{ab}) = \{A, B\}$, $Observers(K_{as}) = \{A, S\}$, and $Observers(K_{bs}) = \{B, S\}$. Also, $\mathcal{W} \in Observers(K)$ means that all principals know K.

4.6.2 Local Sets

Local sets are private to each principal in a protocol specification. In this section we define these sets. Later, we will show how they are used in the actual specification and analysis of a protocol. For each principal, P_i, we define the following sets:

Possession Set(P_i) This set contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. $POSS(P_i) = \{poss_1, poss_2, \cdots, poss_n\}$. $poss_i$ contains two fields: the actual data and the origin of the data.[2]

Belief Set(P_i) This set contains all the beliefs held by a principal. This includes the belief that the keys it holds between itself and other principals are good, beliefs about jurisdiction, beliefs about freshness, and beliefs about the possessions of other principals. $BEL(P_i) = \{bel_1, bel_2, \cdots, bel_n\}$.

Opaque(P_i) This set contains candidates to be added to the seen set. It is used by the `Update` function. The set contains plaintext message parts and a list of the associated keys needed to see them.

Seen(P_i) This set contains plaintext message parts that P_i sees from messages sent across the network. The seen sets collectively contain the same information as the observers sets.

Haskeys(P_i) This set contains keys that P_i sees either because they are in the initial possession set, or because they appear in a message sent across the network and are added to P_i's seen set.

Behavior List(P_i) This item is a list rather than a set because the elements are ordered. $BL = \{AL, bvr_1, bvr_2, \cdots, bvr_n\}$. AL is an action list as will be defined below.

Figure 4.2 shows the structure of BL. The first element of BL, is an action list. The remaining elements, bvr_i, are pairs, $(Mess, AL)$ consisting of a message operation, $Mess$, and an action list, AL.

There are two types of message operations; a message operation is one member of the set $\{Send(P_j, msg),\ Receive(P_j, msg)\}$. $Send(P_j, msg)$ means that P_i sends the message, msg to P_j. Similarly, $Receive(P_j, msg)$ means that P_i receives

[2]Note that the second field represents whether or not P_i generated the data, or who sent it to P_i. It does *not* represent who originated the data.

BEHAVIOR LIST

$action_1, action_2, action_3, \cdots$

Message operation

$action_1, action_2, action_3, \cdots$

Message operation

$action_1, action_2, action_3, \cdots$

Message operation

$action_1, action_2, action_3, \cdots$

$\cdots$

END OF LIST

Figure 4.2: **The Structure of A Behavior List** The list contains a list of actions, followed by a list of pairs, (message operation, action list). After each action, any relevant inference rules are applied.

message msg from P_j. In this case, msg will be marked as coming from P_j and added to $POSS(P_i)$.

In a *send* operation, msg contains the information transmitted. In a *receive* operation msg contains the fields of the expected message. This represents P_i's expectation about the structure of the message. This is similar to the notion of recognizability of the GNY logic [23].

An action list is an ordered list of zero or more actions that are performed by P_i. Actions consist of operations such as encryption and decryption, deletion of information, application of functions, and the decision whether to abort the protocol. They are covered in more detail in section 4.6.5. Every action is followed by a check of the inference rules. If the conditions of a rule are satisfied as a result of the action, then it is applied. These rules are used to update the belief sets of the principals.

Action lists play an important role in protocol specification. Previous approaches to cryptographic protocol analysis take the actions of the principals for granted. Operations such as encryption and decryption are implicit. Our method makes every action explicit, including verification that the operations completed successfully, and an abort in case they did not. This method is a better model of protocol

execution in a real system than previous approaches because all of the actions are included as part of the specification instead of implicitly assumed.

4.6.3 The Trust Matrix

Our method does not require that any assumptions be made about trust between principals. Instead, the protocol designer explicitly specifies the trust relationship between every pair of principals. We define the matrix, TRUST:

$$TRUST[i,j] = \begin{cases} 1 & \text{if } P_i \text{ trusts } P_j \\ 0 & \text{if } P_i \text{ does not trust } P_j \end{cases}$$

The rows and columns enumerate the principals in P. Obviously, when $i = j$, TRUST[i,j] = 1. P_i trusts P_j means that P_i behaves as though P_j will follow the protocol. We give an example of this using a nonmonotonic protocol.

Say that A believes that B possesses X. Now say that the protocol requires that B forget X. As both A and B know the protocol, B should now remove the belief that B possesses X from its belief set. However, if A does not trust B, then he cannot be sure that B actually no longer possesses X. In the actions described below, we stipulate the condition that A trusts B before removing a belief about the possessions of B.

4.6.4 A Word About Nonces

Message freshness can be guaranteed only with time-stamps and nonces. Conceptually, a nonce is a large random number whose purpose is to link a challenge and a response. If A sends a nonce, N_a, to S, then any message including $f(N_a)$, for some function f, and encrypted under K_{as}, is assumed to be fresh if and only if the following conditions are satisfied:

1. No previous message containing $f(N_a)$ has been received.

2. K_{as} is fresh. That is, we assume K_{as} is known only to A and S.

Our method uses inference rules to propagate belief about freshness. In section 4.6.5, we introduce a new construct, LINK(N_a) to link a response to a challenge. When a principal generates a nonce, N_a, the formula LINK(N_a) is added to

his belief set. When a message is received containing, N_a, the LINK item is removed from the belief set, and all parts of that message are labeled as being fresh. A reply to the challenge can be accepted only once. If that message were to be received again, the absence of the LINK item in the belief set would hinder the conclusion that this message is fresh. In fact, this is how our analysis technique exposes the weaknesses in protocols vulnerable to replay attacks. Our analysis of the Needham and Schroeder protocol (Section 4.7.1) gives an example of this.

In our model, the only purpose for a nonce is to link a single challenge to a unique response. Therefore, we require that a nonce be used only once. Existing protocols that link a nonce to messages from several principals can easily be modified to meet this requirement.

4.6.5 Actions

Actions describe how a principal constructs messages, encrypts and decrypts data, computes functions, aborts a protocol, and performs any other operation. The action lists that precede and follow message operations in a principal's behavior list determine sequence of events performed by the principal during a protocol run. As demonstrated below, some of the actions replace inference rules in the BAN logic, and others explicitly represent operations that were taken for granted in previous approaches.

In this section we define the actions used in our method, and the following section presents and discusses the inference rules. Our method requires some new notation and dispenses with some previous constructs. As will be shown, the *said*, *sees*, *controls*, and $Q \overset{K}{\leftrightarrow} P$, constructs of the BAN logic are not needed. The new definitions follow:

X contains Y means that Y appears as a submessage of the message X, more formally, for some (possibly null) x_1, x_2, $X = x_1 \cdot Y \cdot x_2$.[3] It is always the case that X contains X.

S := f(S) represents assignment. The value of S is replaced by the value of the function f applied to S.

[3]We adopt the usual convention of $\cdot$ for concatenation.

X from P means that X is labeled as having been received from P. This will also be true if P generated X.

LINK(N_a) is used to link challenges and responses. This formula is added to the belief set of a principal who generates the nonce N_a, and allows only one subsequently received message to contain the nonce N_a. After such a message is received, the formula is removed from the belief set.

With these new definitions, we now define the actions for a given principal, P_i. Although not specified in the definitions, we assume that *from* labels are inherited in operations. For example, if $\{X\}_k$ is from Q, and is in $POSS(P_i)$, and this is decrypted, then X is also labeled "from Q" when it is added to $POSS(P_i)$.

1. Encrypt(X, k)

condition: $X, k \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \cup \{\{X\}_k\}$

description: This action is used when a principal encrypts data. If P_i possesses X and knows k then he can possess $\{X\}_k$.

2. Decrypt($\{X\}_k, k$)

condition: $\{X\}_k, k \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \cup \{X\}$

description: This action is used when a principal decrypts data. If P_i possesses X, encrypted under k, and P_i knows k, then P_i can possess X.

3. Generate-nonce(N)

result: $POSS(P_i) := POSS(P_i) \cup \{N\}$,
$BEL(P_i) := BEL(P_i) \cup LINK(N)$

description: This action is used when a principal generates a nonce to link a challenge and a response. LINK(N) is removed from $BEL(P_i)$ when the response is received. This is used to determine freshness.

4. Generate-secret(s)

result: $S := S \cup \{s\}$, $Observers(s) = \{P_i\}$, $POSS(P_i) := POSS(P_i) \cup \{s \bowtie P_i\}$,
$BEL(P_i) := BEL(P_i) \cup \#(s)$

description: This action is used when a principal generates a secret data item, such as a key. A new secret, s, is added to S, and the *Observers* and possession sets are updated.

5. **Concat**$(X_1, X_2, \cdots, X_n)$

condition: $X_1, X_2, \cdots, X_n \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \cup$
$\{X_1 \cdot X_2 \cdots X_n\}$

description: This action is used when a principal constructs a message, X, out of submessages $X_1, X_2, \cdots, X_n$.

6. **Split**(X)

condition: X *contains* $x_1 \cdot x_2 \cdots x_n$,
$X \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \cup \{x_1, x_2, \cdots, x_n\}$

description: This action is used to break a message into its components. Split is the opposite of concatenation.

7. **Forget**(X)

condition: $X \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \Leftrightarrow \{X\}, \forall\, P_j \in P$ **if** $TRUST[j,i] = 1$ **then**
$BEL(P_j) := BEL(P_j) \Leftrightarrow \{X \in POSS(P_i)\}$

description: This action is used when P_i no longer is in possession of X. All principals who trust P_i believe that P_i no longer possesses X.

8. **Forget-secret**(s)

condition: $s \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \Leftrightarrow \{s\}$,
$\forall\, P_j \in P$ **if** $TRUST[j,i] = 1$ **then**
$BEL(P_j) := BEL(P_j) \Leftrightarrow \{s \in POSS(P_i)\}$

description: This action is used when P_i no longer knows the secret s. All principals who trust P_i believe that P_i no longer possesses s.

9. Apply(f, X)

condition: $f, X \in POSS(P_i)$

result: $POSS(P_i) := POSS(P_i) \cup \{f(X)\}$

description: This action is used when P_i applies the function f to X. After the application, P_i possesses $f(X)$.

10. Check-freshness(X)

condition: $X \in POSS(P_i)$, X has not expired

result: $BEL(P_i) := BEL(P_i) \cup \{\#(X)\}$

description: This action is used to verify that time-stamp X is fresh.

11. Abort

condition: Protocol run is illegal

result: Analysis reports failure

description: This could happen under various circumstances where there is an inconsistency or other flaw in the protocol specification.

The difference between actions such as *generate* and actions such as *generate-secret* is that items generated as secrets are expected to be sent encrypted, and others are expected to be transmitted in the clear at some point. Many protocols send challenges in the clear; therefore, there is no need to maintain these items as secrets.

The actions described above are used to control the knowledge and possessions of the principals in a protocol. Except for *Check-freshness* and *abort*, all of the actions modify the possession or *Observers* sets. Inference rules are used to modify the belief sets.

The difference between actions and inference rules is that actions are explicitly specified as a part of the protocol. Rules, however, are used to reason about the beliefs of principals as the protocol executes. The protocol builder does not explicitly state how belief evolves in a protocol, but rather, states the inference rules that will mechanically control the propagation of belief.

4.6.6 The Update Function

Before discussing inference rules, we define an important function for processing *send* message operations. When a principal, P_i sends a message to P_j on the network, any principal can read it. In our threat model, we view any message as being broadcast and available to all. As pointed out by Nessett, the BAN logic does not deal with protocols in which, for example, a principal publishes a secret key [50]. This is discussed in Section 4.7.3. The purpose of the `Update` function is to update the *Observers* sets of all secrets that are sent on the network.

We give an algorithm for updating the *Seen* sets for each principal. Updating the *Observers* sets, given these, is simple.

Notation

The cleartext messages in a message are numbered m_i. For example, in the message

$$\{x_1, \{x_2, k_3\}_{k_1}, x_4, x_2\}_{k_2}$$

$m_1 = x_1$, $m_2 = x_2$, $m_3 = k_3$, $m_4 = x_4$, and $m_5 = x_2$. $keys[m_i]$ is a set containing the keys appearing in m_i. In our example, $keys[m_3] = \{k_3\}$, and $keys[m_1] = \phi$. Finally, $needs[m_i]$ is the set of keys needed to see m_i. In our example, $needs[m_2] = \{k_1, k_2\}$, and $needs[m_4] = \{k_2\}$. These sets can be obtained by reading the keys used to encrypt a plaintext submessage from the inside out.

The Algorithm

When a message appears, each principal, P, inspects it, decrypts it insofar as possible, and accumulates any secrets found within.[4]

$$\forall \text{ interior } m_i \tag{4.1}$$

$$opaque(P) \mathrel{+}= m_i \tag{4.2}$$

$$\text{While changes occur to } haskeys(P) \tag{4.3}$$

$$\forall m_i \in opaque(P) \tag{4.4}$$

[4]The notation we use in this algorithm is similar to that of the C programming language. $X \mathrel{+}= Y$ is the same as $X := X + Y$.

$$\text{if } needs[m_i] \subseteq haskeys(P) \tag{4.5}$$

$$opaque(P) \Leftrightarrow= \ m_i \tag{4.6}$$

$$seen(P) \mathrel{+}= m_i \tag{4.7}$$

$$haskeys(P) \mathrel{+}= keys[m_i] \tag{4.8}$$

At each iteration, either keys are added to $haskeys(P)$ or the algorithm halts. Because the number of keys is finite, it will eventually halt.

Lemma:

For a principal, P, m_i is added to $seen(P)$ if and only if P eventually sees all of the keys needed to decrypt m_i. In other words, at some point in the execution of the algorithm, $seen(P) \mathrel{+}= m_i \Longleftrightarrow needs[m_i] \subseteq haskeys(P)$.

Proof: by induction on the structure of messages. A message, M, is constructed via a combination of only three functions, *Encrypt(X,k), Concat(X,Y),* and *Apply(f,X)*, where X and Y are submessages, k is a key, and f is a function. *Apply* does not affect the algorithm.

Basis: If M consists of only a plaintext message, m_i, then $needs[i] = \phi$ and thus $needs[i] \subseteq haskeys(P)$. Therefore, $seen(P) \mathrel{+}= m_i$. The lemma holds.

Inductive Hypothesis: Assume that if M is a message containing at most n applications of *Encrypt* and *Concat*, then for each $m_i \in M$, $seen(P) \mathrel{+}= m_i \Longleftrightarrow needs[m_i] \subseteq haskeys(P)$.

Need to prove that if M' is a message containing at most $n+1$ applications of *Encrypt* and *Concat*, then for all $m'_i \in M'$, $seen(P) \mathrel{+}= m'_i \Longleftrightarrow needs[m'_i] \subseteq haskeys(P)$.

Observe that M' must be of the form *Encrypt(X,k)* or *Concat(X,Y)* Where X and Y contains at most n applications of *Encrypt* and *Concat.*

case 1

M' is of the form *Encrypt(X,k).* In this case, if $k \in haskeys(P)$, then for each $m'_i \in M'$, and for each $m_i \in X$, $needs[m'_i] = needs[m_i] \cup \{k\}$, and $seen(P) \mathrel{+}= m'_i$ **iff** $seen(P) \mathrel{+}= m_i$. If $k \notin haskeys(P)$, then for all $m_i \in X, needs[m_i] \not\subseteq haskeys(P)$ because $k \in needs[m_i]$, and m_i is not added to $seen(P)$. Thus, the lemma holds for this case.

case 2

M' is of the form *Concat(X,Y)*. In this case, the lemma holds for X and Y by the inductive hypothesis, and *Concat* does not effect the *needs* and *haskeys* sets. Therefore, the lemma holds. ∎

4.6.7 Inference Rules

When using our method, the protocol developer must choose from the eleven actions defined above. In addition, we define inference rules, such as the nonce verification rule of the BAN logic, that are triggered whenever they apply. In our method, rules are specified a bit differently.

The nonce verification rule as defined by Burrows *et al.* [9] is not entirely intuitive. Their rule is:

$$\frac{\text{P believes } \#(X),\ \text{P believes Q said X}}{\text{P believes Q believes X}}$$

It is not clear what it means for Q to believe X, if X is a nonce. In our method, if P received X from Q, then $X \in POSS(P_i)$, and X will be labeled as being from Q. We define the nonce verification rule as follows:

$$\frac{\begin{array}{c}(X \in POSS(Q)) \in BEL(P),\\ \#(X) \in BEL(P),\ X \text{ from } Q \in POSS(P)\end{array}}{BEL(P) := BEL(P) \cup \{Q \text{ believes } \#(X)\}}$$

The first condition, $(X \in POSS(Q)) \in BEL(P)$ is necessary to authenticate X as coming from Q. This condition can only be true after the message meaning rule (defined below) is applied. Thus, P can establish that Q believes that X is fresh, and this fact is added to P's belief set, $BEL(P)$. Notice that rules are used to propagate belief during a protocol run, whereas actions deal with knowledge.

The message meaning rule is defined in BAN as:

$$\frac{\text{P believes Q} \overset{K}{\leftrightarrow} \text{P, P sees } \{X\}_K}{\text{P believes Q said X}}$$

This rule states that if P believes that Q and P share a secret key, K, and P sees X, encrypted under K, and P did not encrypt X under K, then P believes that Q once said X. This implies that knowing a principal shares a secret key with another principal is enough to guarantee that any message encrypted under that key was

sent by that principal. In our rule, this requirement is made explicit. We define the message meaning rule as follows:

$$\frac{\{X\}_k \text{ from } Q \in POSS(P),\ k \in POSS(P)}{BEL(P) := BEL(P) \cup \{X \in POSS(Q)\}}$$

This rule states that if $\{X\}_k$ was received by P, from Q, and both P and Q know the key k, then P believes that Q possesses X. This is possible because messages are labeled with their origin when they are added to the possession set. When principal P applies action 2 (decrypt), X will be in his possession set.

Another rule is needed to reason about the freshness of submessages:[5]

$$\frac{\begin{array}{c}\#(x_1) \in BEL(P), \\ \{X \text{ contains } x_1, X \text{ contains } x_2\} \subseteq POSS(P)\end{array}}{BEL(P) := BEL(P) \cup \#(x_2)}$$

This rule states that if P believes that x_1 is fresh, and P possesses a formula containing both x_1 and x_2, then P believes that x_2 is fresh. This rule reflects the fact that any part of a message which contains something fresh, is fresh.

The following rule is the most important inference rule for reasoning about freshness in a protocol. The only way for a message to be fresh is for it to contain a valid time-stamp or a nonce that has never previously been used in a response. The rule for determining freshness using a nonce is the linkage rule:

$$\frac{\begin{array}{c}\#(k) \in BEL(P), k \in POSS(P), \\ LINK(N_a) \in BEL(P), X \text{ contains } f(N_a), \\ X \text{ contains } x_1, \{X\}_k \text{ from } Q \in POSS(P)\end{array}}{BEL(P) := (BEL(P) \Leftrightarrow LINK(N_a)) \cup \{\#(x_1)\}}$$

This rule is simpler than its unfortunate length makes it appear. It is the only rule that can be used to add information about the freshness of an item which is not known to contain fresh submessages, to a principal's belief set.

The linkage rule states that the submessages of a message X are believed to be fresh under certain conditions. If LINK(N_a) is in P's belief set, then the nonce N_a has not been used before. This is the first condition. If the rule is applied successfully, the LINK item is removed. So the rule could not fire again for the same nonce. Other

[5] In this rule, it is assumed that x_1 and x_2 share an outer encryption to prevent a malicious intruder from substituting one of them.

conditions state that the nonce N_a must be sealed under a key that is fresh, and must be available to P.

The rule states that message X must contain $f(N_a)$ to represent the fact that sometimes a function of a nonce, rather than the actual nonce is used to respond to a challenge. The net result of applying this rule is that any submessage of a valid response to a challenge is believed to be fresh by the recipient of the response. Also, there is a guarantee that any replay of a valid response will not result in a principal believing that the submessages are fresh.

4.6.8 How It All Works

The analysis of a protocol proceeds from the specification. It may be a partial specification or a whole one. Figure 4.3 shows the flow of control in the analysis of a protocol with two principals. One action is marked as the first, and this

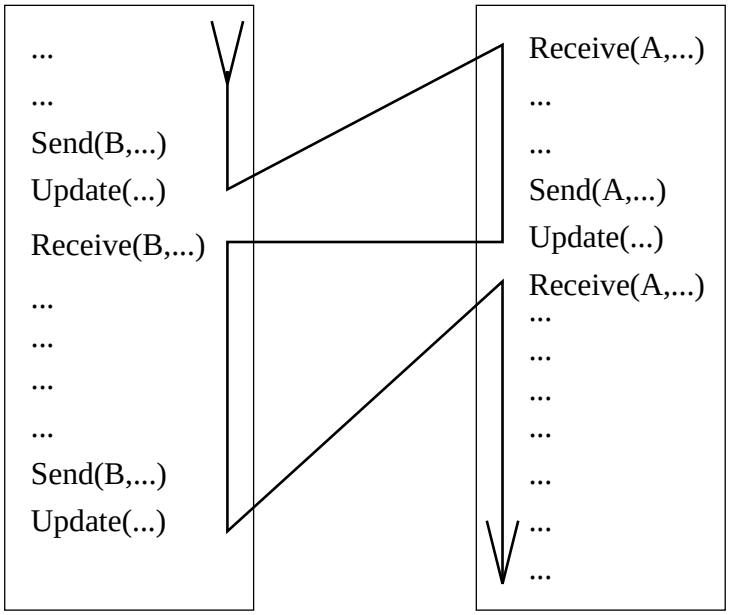

Figure 4.3: **The Flow of Control in Protocol Analysis.** This diagram shows how the analysis proceeds sequentially through the behavior list of two principals in a protocol. After each `Update`, the analysis moves to the next *receive* of the principal specified in the previous *send* operation.

is where the analysis begins. The conditions of the action are tested, and if they are true, the results are applied. Otherwise, the analysis aborts because the specification

is unsound.

After the results are applied, any relevant inference rules are applied. There are no race conditions, where the results of one inference rule negate the preconditions of another, because the inference rules are monotonic. Only actions can result in the deletion of information. Therefore, the order of the rules does not matter. The rules only fire once for any given set of preconditions, and as the number of data elements is finite, this process will eventually halt. At this point, the current action is marked as seen.

The analysis then moves to the next action in the same action list. If the action is a *send* message operation, then the following action must be an `Update` function call. After the *Observers* sets are updated, the analysis moves to the first unseen *receive* action in the action list of the principal specified in the *send*.

When `Update` is called, the *Observers* sets are updated according to the algorithm described in Section 4.6.6. If there is a *send* that is not followed by a call to `Update`, then the specification is not sound.

After the analysis completes, the protocol designer checks if all of the actions are seen. If they are not, then there is a problem with the protocol. Flaws can be detected at any point in the analysis. There are many different ways to use this technique to discover them. For example, the *Observers* set of a secret may contain the wrong principals. If the analysis does not conclude that a data item is fresh, then a replay attack is possible. In addition, an intruder's moves can be modeled as a participant in the protocol, which can be checked for known attacks.

4.7 Examples

The best way to explain how a protocol is analyzed using our method is by example. In section 4.7.1, the Needham and Schroeder protocol is specified, and we step through the analysis. In section 4.7.2, we apply our method to the *khat* protocol.

4.7.1 Needham and Schroeder

First we specify the protocol, and then we show how our method can be used to analyze it. We demonstrate that the known flaw in the protocol exists; principal B cannot achieve the belief that the session key with A is fresh. Then, we

show how the addition of two messages, as proposed by Needham and Schroeder (in a later paper [49]) to solve the problem, allows B to achieve the desired belief.

4.7.1.1 The Specification

The Needham and Schroeder protocol assumes that both A and B trust S. So, the trust matrix contains 1s in the appropriate spots to represent this. The trust between A and B is irrelevant, and so the matrix values do not matter.

First we define the global sets. $P = \{S, B, A\}$. A is marked as the initiator of the protocol. R contains the rules defined in Section 4.6.7. $S = \{K_{as}, K_{bs}\}$. Each of these secret keys has an *Observers* set. $Observers(K_{as}) = \{A, S\}$ and $Observers(K_{bs}) = \{B, S\}$. Some of these sets will change once the analysis begins.

Next, we define the initial values of the local sets. Notice that initially, principals believe in the freshness of the key they share with the server, S. Similarly, the server believes in the freshness of its shared secret with each principal. Also, $P \bowtie K_{pq}$ represents the key, K_{pq} and the fact that it is to be used for communicating with principal P. In this protocol, the function f subtracts one from its argument.

Principal A

$POSS(A) = \{K_{as}\}$
$BEL(A) = \{\#(K_{as})\}$
$BL(A) =$

- Generate-nonce(N_a)
  Concat(A, B, N_a)
  Send($S, \{A \cdot B \cdot N_a\}$)
  Update($\{A \cdot B \cdot N_a\}$)
  Receive($S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)
  Decrypt($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, K_{as}$)
  Split($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$)
  Send($B, \{A \bowtie K_{ab}\}_{K_{bs}}$)
  Update($\{A \bowtie K_{ab}\}_{K_{bs}}$)
  Receive($B, \{N_b\}_{K_{ab}}$)
  Decrypt($\{N_b\}_{K_{ab}}, K_{ab}$)

Send(B, Encrypt(Apply(f, N_b), K_{ab}))
Update($\{f(N_b)\}_{K_{ab}}$)

Principal B

$POSS(B) = \{K_{bs}\}$
$BEL(B) = \{\#(K_{bs})\}$
$BL(B) =$
Receive($A, \{A \bowtie K_{ab}\}_{K_{bs}}$)
Decrypt($\{A \bowtie K_{ab}\}_{K_{bs}}, K_{bs}$)
Generate-nonce(N_b)
Send(A,Encrypt(N_b, K_{ab}))
Update($\{N_b\}_{K_{ab}}$)
Receive($A, \{f(N_b)\}_{K_{ab}}$)
Decrypt($\{f(N_b)\}_{K_{ab}}, K_{ab}$)

Principal S

$POSS(S) = \{K_{as}, K_{bs}\}$
$BEL(S) = \{\#(K_{as}), \#(K_{bs})\}$
$BL(S) =$
Receive($A, \{A \cdot B \cdot N_a\}$)
Split($\{A \cdot B \cdot N_a\}$)
Generate-secret(K_{ab})
Send(A,Encrypt(Concat($N_a, B \bowtie K_{ab}$,
Encrypt($A \bowtie K_{ab}, K_{bs}$)), K_{as}))
Update($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)

Once the protocol has been specified, the analysis begins. However, the analysis technique described here can be used to test the protocol as it is being developed.

The first action in $BL(A)$ is marked with a $\bullet$ because A is the initiator of the protocol. For each action, its condition is tested. If it does not hold, the protocol analysis is aborted, and the specification is infeasible. If the condition holds, then the result is applied and the required sets are updated. Next, the inference rules are

examined to see if any apply. Finally, the action is marked with a ∘ to show that it has been successful, and the mark, •, is moved to the next action.

Every `Send` action is followed by an `Update` action. The `Send` action specifies to whom the message is sent. After an `Update` action, the mark moves to the first `Receive` action with no ∘ of the principal identified in the corresponding *Send* action.

4.7.1.2 The Analysis

The first four actions in $BL(A)$ are executed resulting in new members of the sets $POSS(A)$ and $BEL(A)$. Also, the `Update` action causes $Observers(N_a) = \mathcal{W}$. So far, no inference rules can be applied.

$POSS(A) = \{K_{as}, N_a, \{A \cdot B \cdot N_a\}\}$
$BEL(A) = \{\#(K_{as}), \text{LINK}(N_a)\}$
$BL(A) =$
∘ Generate-nonce(N_a)
∘ Concat(A, B, N_a)
∘ Send($S, \{A \cdot B \cdot N_a\}$)
∘ Update($\{A \cdot B \cdot N_a\}$)
Receive($S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)
...

After the `Update` action, the next action to be executed is in S's behavior list because the `Send` action specifies S.

• Receive($A, \{A \cdot B \cdot N_a\}$)

The five actions of $BL(S)$ are executed. There are still no relevant inference rules. The set S now contains $\{K_{as}, K_{bs}, K_{ab}\}$. After applying the `Update` function (the last action of principal S), $Observers(K_{ab}) = \{S, A\}$ because $A \in Observers(K_{as})$. The term $\{A \bowtie K_{ab}\}_{K_{bs}}$ does not cause B to be added to the *Observers* set of K_{ab} because B is not a member of $Observers(K_{as})$, and so B is not in $Observers(K_{as}) \cap Observers(K_{bs})$. The possession set contains subparts of messages that were built as the messages were constructed, but we omit these here for space consideration as they do not contribute in any way to the analysis. The new values of S's local sets are:

$POSS(S) = \{K_{as}, K_{bs}, \{A \cdot B \cdot N_a\}$ from A,
$K_{ab}, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}\}$
$BEL(S) = \{\#(K_{as}), \#(K_{bs}), \#(K_{ab})\}$
$BL(S) =$
$\circ$ Receive$(A, \{A \cdot B \cdot N_a\})$
$\circ$ Split$(\{A \cdot B \cdot N_a\})$
$\circ$ Generate-secret(K_{ab})
$\circ$ Send(A,Encrypt(Concat($N_a, B \bowtie K_{ab}$,
Encrypt$(A \bowtie K_{ab}, K_{bs})), K_{as}))$
$\circ$ Update$(\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}})$

The next action is in A's BL.

- Receive$(S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}})$

The term $\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$ is added to POSS(A). The next action to be executed is:

- Decrypt$(\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, K_{as})$

This will add the term $\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$ to $POSS(A)$. The next action, `Split` will add the individual components too.

At this point, the conditions for the linkage rule are satisfied. We take X to be the term $\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$ that was just added to $POSS(A)$. The reader can verify that the following are all true:

1. $\#(K_{as}) \in BEL(A)$
2. $A \in Observers(K_{as})$
3. LINK$(N_a) \in BEL(A)$
4. X *contains* $g(N_a)$, where g is the identity function
5. $\{X\}_{K_{as}}$ from $S \in POSS(A)$

Once the linkage rule is applied, the freshness of each subpart of X is added to belief set of A. Also, the LINK formula is removed from the belief set so that the nonce N_a cannot be used again.

At this point, the global sets have not changed. The sets for principal A are as follows (We omit items in the possession and belief sets, such as large concatenated messages, that serve no further purpose.):

Principal A

$POSS(A) = \{K_{as}, N_a, B \bowtie K_{ab}, \{A \bowtie K_{ab}\}_{K_{bs}}\}$
$BEL(A) = \{\#(K_{as}), \#(K_{ab}), \#(\{A \bowtie K_{ab}\}_{K_{bs}})\}$
$BL(A) =$
∘ Generate-nonce(N_a)
∘ Concat(A, B, N_a)
∘ Send($S, \{A \cdot B \cdot N_a\}$)
∘ Update($\{A \cdot B \cdot N_a\}$)
∘ Receive($S, \{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}$)
∘ Decrypt($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}_{K_{as}}, K_{as}$)
∘ Split($\{N_a \cdot B \bowtie K_{ab} \cdot \{A \bowtie K_{ab}\}_{K_{bs}}\}$)
• Send($B, \{A \bowtie K_{ab}\}_{K_{bs}}$)
Update($\{A \bowtie K_{ab}\}_{K_{bs}}$)
Receive($B, \{N_b\}_{K_{ab}}$)
Decrypt($\{N_b\}_{K_{ab}}, K_{ab}$)
Send(B, Encrypt(Apply(f, N_b), K_{ab}))
Update($\{f(N_b)\}_{K_{ab}}$)

The `Send` and `Update` actions in A's behavior list are executed next. The `Update` function adds B to $Observers(K_{ab})$. The next action to be executed is in B's behavior list, as specified by the last `Send` action.

• Receive($A, \{A \bowtie K_{ab}\}_{K_{bs}}$)

The next action on B's behavior list is:

• Decrypt($\{A \bowtie K_{ab}\}_{K_{bs}}, K_{bs}$)

After this action is executed, $\{A \bowtie K_{ab}\}$ is added to $POSS(B)$. However, the linkage rule does not apply because there is no LINK statement in $BEL(B)$. Thus, *B cannot conclude that K_{ab} is fresh!*

In fact, when B receives $\{f(N_b)\}_{K_{ab}}$ from A, it cannot apply the linkage rule because one of the conditions is that K_{ab} is fresh. For the remainder of the protocol, B can never conclude that anything received under K_{ab} is fresh.

This is the same flaw discovered in the Needham and Schroeder protocol by Denning and Sacco [17]. We apply Needham and Schroeder's fix [49] by adding several actions to the beginning of the behavior lists of A and B. To $BL(A)$, we add the actions:

Send$(B, \{A\})$
Update$(\{A\})$
Receive$(B, \{A, J\}_{K_{bs}})$
Decrypt$(\{A \cdot J\}_{K_{bs}}, K_{bs})$
Split$(\{A \cdot J\})$

And to $BL(B)$, we add the actions:

Receive$(A, \{A\})$
Generate-nonce(J)
Send(A, Encrypt(Concat$(A, J), K_{bs}$))
Update$(\{A \cdot J\}_{K_{bs}})$

Then, A will include J in the original message to S, and S will include it in $\{A \bowtie K_{ab}\}_{K_{bs}}$ that gets forwarded to B.

It is clear that when B generates J, a LINK statement is added to $BEL(B)$. When B receives the message containing K_{ab} from A, it will be able to conclude $\#(K_{ab})$. Also, because $Observers(K_{bs}) = \{B, S\}$ throughout the protocol, no intruder could generate or modify the forwarded message from A to B that is sealed under K_{bs}.

Thus, our analysis reveals no flaws in the revised Needham and Schroeder protocol.

4.7.2 KHAT

Our method for analyzing cryptographic protocols does not include temporal reasoning. Thus, we specify and analyze the two phases of the *khat* protocol separately.

One advantage of our method is that the *khat* protocol can be specified in the same manner as the Needham and Schroeder protocol; we specify all the global and local sets. The behavior lists will contain actions that precisely describe the protocol. Section 4.5 shows that the previous method of listing the messages between principals is inadequate as a specification technique.

Our analysis reveals a significant flaw in the *khat* protocol. We provide a fix to the protocol, and use the analysis to demonstrate that the flaw no longer exists.

4.7.2.1 The Specification

The *khat* protocol involves two principals: the client (C) and the server (S). In this protocol, the trust matrix must reflect the fact that they trust each other. The client trusts the server to issue valid tickets, and the server trusts the client to forget the information specified in the protocol. If the TRUST$[i, j]$, where i is the server and j is the client, is not 1, then the server will not believe that the client no longer possesses information which should be forgotten. Thus, a fundamental assumption of the protocol is identified.

When phase I begins, we assume that a secure channel has been established using the Kerberos ticket for the *khat* service. Thus, K is the session key between C and S, and $Observers(K) = \{C, S\}$. $P = \{S, C\}$, C is marked as the initiator, and $S = \{K\}$. In this specification, SF represents the spool file for the user's job. The local sets are now defined:

Client

$POSS(C) = \{K\}$
$BEL(C) = \{\#(K)\}$
$BL(C) =$

- Generate-secret(SF)
  Generate-secret(N)
  Encrypt(K,N)
  Send(S,Encrypt(Concat(SF, N), K))
  Update($\{SF \cdot N\}_K$)
  Forget-secret(N)

Forget-secret(SF)[6]

Phase II

Receive($S, \{N \cdot \{SF \cdot TGT_C\}_K\}$)

Split($\{N \cdot \{SF\}_K\}$)

Decrypt($\{K\}_N, N$)

Decrypt($\{SF \cdot TGT_C\}_K, K$)

Split($\{SF \cdot TGT_C\}$)

Check-Freshness(TGT_C)

Server

$POSS(S) = \{K\}$

$BEL(S) = \{\#(K)\}$

$BL(S) =$

Receive($C, \{SF \cdot N\}_K$)

Decrypt($\{SF \cdot N\}_K, K$)

Split($\{SF \cdot N\}$)

Phase II

Generate-secret(TGT_C)

Send(Concat(N,Encrypt(Concat(SF, TGT_C), K)))

Update($\{N \cdot \{SF \cdot TGT_C\}_K\}$)

4.7.2.2 The Analysis

We begin our analysis with Phase I of the protocol. After the analysis reaches the first `Forget-secret` statement, the local sets are as follows (once again we omit some encrypted items in the possession and belief sets that don't contribute to the analysis, for the sake of clarity):

---

[6]For completeness sake, we should also specify to forget $\{SF \cdot N\}_K$ and other formulas that are added to $POSS(C)$ by Concat and Encrypt, but we will omit these from the BL for clarity. They would be included in an actual specification (and their existence helped the authors discover a bug in the actual *khat* implementation).

Client

$POSS(C) = \{K, SF, N, \{K\}_N, \}$

$BEL(C) = \{\#(K), \#(N), \#(SF)\}$

$BL(C) =$

- Generate-secret(SF)
- Generate-secret(N)
- Encrypt(K,N)
- Send(S,Encrypt(Concat(SF, N), K))
- Update($\{SF \cdot N\}_K$)
- Forget-secret(N)

  Forget-secret(SF)

Server

$POSS(S) = \{K, N, SF\}$

$BEL(S) = \{\#(K)\}$

$BL(S) =$

- Receive($C, \{SF \cdot N\}_K$)
- Decrypt($\{SF \cdot N\}_K, K$)
- Split($\{SF \cdot N\}$)

Notice that the server cannot conclude $\#(SF)$ or $\#(N)$. This is a serious flaw because an intruder can use a replay attack for the remainder of the session[7] to reschedule the user's job.

To solve this problem, we modify the protocol so that along with the *khat ticket*, the server sends a list of fresh nonces to the client. Each time the user schedules a job, he includes an unused nonce in the message. In the analysis, the server will have a collection of LINK statements in its belief set, and the freshness of N and SF can be guaranteed.

[7] That is, the remaining lifetime of the *khat* ticket from the ticket granting service.

4.7.2.3 The Corrected Protocol

The corrected protocol is as follows:

Client

$POSS(C) = \{K\}$

$BEL(C) = \{\#(K)\}$

$BL(C) =$

Part of ticket granting

Receive(S,$\{N_1, N_2, \cdots N_n\}$)

Split($\{N_1, N_2, \cdots N_n\}$)

Phase I

Generate-secret(SF)

Generate-secret(N)

Encrypt(K,N)

Send(S,Encrypt(Concat(SF, N, N_i[8]), K))

Update($\{SF \cdot N \cdot N_i\}_K$)

Forget-secret(N)

Forget-secret(SF)

Phase II

Receive($S, \{N \cdot \{SF \cdot TGT_C\}_K\}$)

Split($\{N \cdot \{SF\}_K\}$)

Decrypt($\{K\}_N, N$)

Decrypt($\{SF \cdot TGT_C\}_K, K$)

Split($\{SF \cdot TGT_C\}$)

Check-freshness(TGT_C)

Server

$POSS(S) = \{K\}$

$BEL(S) = \{\#(K)\}$

$BL(S) =$

---

[8] N_i is the first unused nonce in the list received from the server.

Part of ticket granting

- Generate-nonce(N_1)

  Generate-nonce(N_2)

  $\cdots$

  Generate-nonce(N_n)

  Send(C,Concat($N_1, N_2, \cdots, N_n$))

  Update($\{N_1, N_2, \cdots N_n\}$)

Phase I

Receive($C, \{SF \cdot N \cdot N_i\}_K$)

Decrypt($\{SF \cdot N \cdot N_i\}_K, K$)

Split($\{SF \cdot N \cdot N_i\}$)

Phase II

Generate-secret(TGT_C)

Send(Concat(N,Encrypt(Concat(SF, TGT_C), K)))

Update($\{N \cdot \{SF \cdot TGT_C\}_K\}$)

Now, after the analysis reaches the first `Forget-secret` statement, $BEL(S)$ contains (among other things) $\#(K)$, $LINK(N_2)$, $\cdots$, $LINK(N_n)$, $\#(SF)$, and $\#(N)$. The flaw described earlier no longer exists. If an intruder attempts to replay the message containing the spool file, the server will recognize that the nonce, N_i has already been used. In the analysis, this is reflected by the absence of $LINK(N_1)$ from $BEL(S)$. The linkage rule cannot be applied in this case. Thus, the server will not conclude that the spool file in the replayed message is fresh, and the protocol will be aborted.

We continue our analysis with the client's actions:

- Forget-secret(N)

  Forget-secret(SF)

After these actions, N and SF are removed from $POSS(C)$. Also, the beliefs that C possesses N and SF are removed from $BEL(S)$ because S trusts C according to the trust matrix. At this point phase I is over. It is clear from the values of the *Observers* sets, which are updated with every `Send` action, that nobody can learn the value of SF from the messages sent. Also, the possession set of C represents what an intruder can learn by compromising the workstation while a job is pending. The only useful

possession is $\{K\}_N$. Of course, without N, this is useless. Because $Observers(N) = \{S\}$, no intruder can gain anything by compromising the workstation before phase II begins.

To preserve space, we include only the most interesting part of the analysis that remains. When phase II begins, the next three actions are the server's.

- Generate-secret(TGT_C)
  Send(Concat(N,Encrypt(Concat(SF, TGT_C), K)))
  Update($\{N \cdot \{SF \cdot TGT_C\}_K\}$)

After the `Update` action, $Observers(N) = \mathcal{W}$. Thus, if an intruder has compromised the workstation and obtained $\{K\}_N$, then the secrecy of K has also been lost. Thus, analysis reveals that once it is time for the job to run, a previous compromise of the workstation results in an insecure session key. This further results in the compromise of the TGT.

Our analysis reveals a new vulnerability in phase II of *khat*. Although the analysis did not mechanically produce this result, use of our technique generated conclusions from which the vulnerability became apparent. In Section 4.8 we discuss how to test a protocol for known weaknesses.

4.7.3 Nessett criticism

In a well known note [50], Nessett criticizes the BAN logic. He presents the following protocol that uses asymmetric keys:

$A \rightarrow B : \{N_a, K_{ab}\}_{K_a{}^{-1}}$

$B \rightarrow A : \{N_b\}_{K_{ab}}$

The problem is that K_{ab} is encrypted under A's private key. Thus, anyone intercepting the first message can decrypt it with the corresponding public key and obtain the session key.

Once the protocol is specified, our analysis immediately reveals the flaw. After the first message is sent, the update function sets $Observers(K_{ab})$ to $\mathcal{W}$ because the *Observers* set of the public key K_a is $\mathcal{W}$. In addition, B does not believe that K_{ab} is fresh.

Interestingly, our analysis also reveals that in addition to its obvious and intended flaw, the Nessett protocol uses nonces improperly.

4.8 Analyzing Known Threats

Our specification and analysis technique can also be used to test a protocol against a known attack. This is done by including the intruder, Z, in the set of principals. $BL(Z)$ contains the actions that the intruder performs. The analysis determines what Z is able to learn during the course of the protocol. The trust matrix can even be used to analyze what happens when Z is actually trusted.

By specifying $BL(Z)$ differently, one can determine whether an intruder could trick a participant into revealing some sensitive information using a given attack. In this sense, a user can interact with the analysis to check a new protocol for given flaws and vulnerabilities. Following is an example of a very simple specification of part of a protocol to illustrate this. The protocol piece is a challenge/response pair, and the example shows how to specify the intruder's role. In this attack, the intruder, Z, uses B to figure out what response to send to A. In this manner, Z fools A into believing that he can encrypt N_a.

Specification without intruder:

Principal A

$POSS(A) = \{N_a, K_{ab}\}$
$BEL(A) = \{\#(K_{ab}), \#(N_a)\}$
$BL(A) =$
  Send$(B, \{N_a\})$
  Update$(\{N_a\})$
  Receive$(B, \{N_a\}_{K_{ab}})$

Principal B

$POSS(B) = \{K_{ab}\}$
$BEL(B) = \{\#(K_{ab})\}$
$BL(B) =$

Receive$(A, \{N_a\})$

Send(A, Encrypt$(\{N_a\}, K_{ab})$

Update$(\{N_a\}_{K_{ab}})$

Specification with intruder:

Principal A

$POSS(A) = \{N_a, K_{ab}\}$

$BEL(A) = \{\#(K_{ab}), \#(N_a)\}$

$BL(A) =$

Send$(Z, \{N_a\})$

Update$(\{N_a\})$

Receive$(Z, \{N_a\}_{K_{ab}})$

Principal B

$POSS(B) = \{K_{ab}\}$

$BEL(B) = \{\#(K_{ab})\}$

$BL(B) =$

Receive$(Z, \{N_a\})$

Send(Z, Encrypt$(\{N_a\}, K_{ab})$

Update$(\{N_a\}_{K_{ab}})$

Principal Z

$POSS(Z) = \{\}$

$BEL(Z) = \{\}$

$BL(Z) =$

Receive$(A, \{N_a\})$

Send$(B, \{N_a\})$

Update$(\{N_a\})$

Receive$(B, \{N_a\}_{K_{ab}})$

Send$(A, \{N_a\}_{K_{ab}})$

Update$(\{N_a\}_{K_{ab}})$

4.9 Conclusions

In this chapter, we introduce a new method for specifying authentication protocols that offers several advantages over existing methods. The method also includes a logical analysis based on the propagation of belief and knowledge. A fundamental assumption in our threat model is that any message in the system is essentially a broadcast.

We specify protocols as a collection of independent processes. This model closely resembles the structure of the actual distributed system in which the protocols are implemented. Our specifications are designed to resemble the actual implementation as much as possible. This eliminates flaws introduced in the process of converting a specification (which may contain no flaws itself) to an actual program.

One weakness of many analysis techniques that require protocol idealization is that flaws in the protocol may not appear in the idealized version. Thus, the analysis is incapable of revealing them. Our method does not require idealization and thus avoids this problem.

We demonstrate that our method can be used to reason about a new class of protocols for which previous approaches are inadequate. We use actions such as `Forget` and `Forget-secret` along with knowledge and belief sets to reason about nonmonotonicity of knowledge in protocols.

The Needham and Schroeder protocol has become a benchmark used by designers of analysis techniques to test their methods. We demonstrate how the known flaw in that protocol is revealed. In addition, we use our method to uncover a new flaw in our *khat* protocol and to discover a vulnerability in phase II of the protocol. Finally, we show that our method easily uncovers flaws in protocols, such as Nessett's, that methods such as BAN cannot detect.

CHAPTER 5

NONMONOTONIC CRYPTOGRAPHIC PROTOCOLS WITH PUBLIC KEYS

5.1 Overview

This chapter presents extensions to the technique presented in the previous chapter for protocols that use asymmetric keys. We introduce new actions and inference rules, as well as slight modifications to the `Update` function. An important observation is that reasoning about the origin of messages is quite different when dealing with asymmetric key protocols.

We also introduce the notion that keys in certificates should be *bound* to the principals receiving them. We extend the technique to meet the binding requirements and show how the flaw in the Denning and Sacco public key protocol, that was discovered by Abadi and Needham, is revealed.

We demonstrate the extended technique using one protocol of our own and the Needham and Schroeder public key protocol. Finally, we introduce and analyze a fix to a known weakness in Needham and Schroeder's protocol using our extended technique.

5.2 Introduction

In the previous chapter, we introduce a technique for specifying and analyzing nonmonotonic cryptographic protocols. The technique is used to demonstrate a known flaw in the Needham and Schroeder protocol [48], and to discover a flaw in the *khat* protocol.

In addition to discovering flaws, the specification and analysis technique offers several advantages over previous methods. The method properly handles protocols that are trivially insecure, such as that of Nessett [50], and allows specification and analysis of protocols that rely on nonmonotonicity of knowledge. In addition, *idealization* and its associated pitfalls [38] are avoided. Finally, the specification leads directly to an implementation.

In this chapter, we extend this technique for protocols that use asymmetric keys. One of the greatest difficulties presented by such protocols is determining the origin of messages. Section 5.7.2 addresses this problem.

We use an example from Abadi and Needham [1] to show how key certificates with unbound keys can result in flaws. We present a sample protocol to demonstrate the extended technique. Then, we uncover a known weakness in the Needham and Schroeder public key protocol [48] to further demonstrate the extensions. We introduce a modified version of Needham and Schroeder that corrects the weakness, and use the extended technique to analyze the new protocol.

The appendices contain information describing the technique. Appendix A gives a complete list of actions. Appendix B contains a description of the sets used for specification and analysis, and Appendix C contains a summary of the inference rules.

5.3 Properties of Asymmetric Keys

The use of asymmetric keys in a cryptosystem is often termed *public key cryptography* [18]. In these systems, users are in possession of a pair of keys. Typically, one is called the private key and is available to only one user. The other is the public key, and it is made available to everyone. The public and private keys are inverses of each other, and there is only one encryption function. If a message is encrypted with the private key, then it can be recovered by reencrypting with the public key, and vice versa.

In our method, we discuss asymmetric keys only in the context of their inverse property. Their use as *public* or *private* keys is a convention that can be specified in a protocol.

We do not assume that public keys are available to all principals. The goal

of a protocol may be to distribute public keys safely, and such an assumption makes it impossible to specify the protocol. Instead, a protocol designer can include such assumptions in the specification.

A protocol designer can specify asymmetric keys as public/private keys by including the initial assumption that one of the keys in the pair is held only by the principal who generated it. As long as that key is not included in any message, it remains the private key. Additional actions can be defined for verifying signatures. We do not include the details here.

Finally, we present some notation. k^+ and k^- are asymmetric keys that are inverses of each other. When dealing with protocols that assign the names *public* and *private* to these, we adopt the convention that k^+ is the public key and k^- is the private key.

5.4 The Problem of Unbound Keys

This section addresses a problem with cryptographic protocols that use asymmetric keys. The problem is first addressed by Abadi and Needham [1]. They discover a flaw in the Denning and Sacco protocol [17] that uses timestamps. The flaw is based on the fact that a session key is sent without the name of the receiving principal. Abadi and Needham give an attack wherein a malicious principal, C, can forward a signed certificate from principal A, to B, and subsequently masquerade as A.

The Denning and Sacco protocol, as presented in simplified form by Abadi and Needham is:

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : CA, CB$
3. $A \rightarrow B : CA, CB, \{\{k, T_a\}_{k_a-}\}_{k_b+}$

In this example, T_a is a timestamp, k is a session key, and CA and CB are signed certificates from the trusted server, S, that contains the public keys of A and B respectively; their format is not important for this example. The goal of the protocol is to safely distribute the session key, k, to A and B.

The attack by malicious principal, C, given by the authors is as follows:

1. $A \rightarrow S : A, C$
2. $S \rightarrow A : CA, CC$
3. $A \rightarrow C : CA, CC, \{\{k, T_a\}_{k_a-}\}_{k_c+}$
4. $C \rightarrow S : C, B$
5. $S \rightarrow C : CC, CB$
6. $C \rightarrow B : CA, CB, \{\{k, T_a\}_{k_a-}\}_{k_b+}$

In this attack, A initiates the protocol with C. Then, C is in possession of $\{k, T_a\}_{k_a-}$, which C can decrypt to recover k. Then, C constructs $\{\{k, T_a\}_{k_a-}\}_{k_b+}$ to send message 6. Now B is convinced that k is a fresh key to be used only with principal A.

This attack is successful because k is not bound to any principal. We introduce the notion of binding an encryption key to a principal. We represent the key, k, along with its binding to principal B as $k \bowtie B$. If message three in the Denning and Sacco protocol is replaced with

3. $A \rightarrow B : CA, CB, \{\{k \bowtie B, T_a\}_{k_a-}\}_{k_b+}$

the attack fails.

We now define the notion of a bound key. In general, a symmetric key is bound to each principal that receives it in a message. In the above example, A binds k to B before sending message 3. Important goals in any symmetric key protocol are $(k \bowtie A) \in POSS(A)$, $(k \bowtie B) \in POSS(B)$, etc. If A receives $(k \bowtie B)$ in a message, then the protocol should abort.[1]

We also define the notion of a bound key for asymmetric key systems. An asymmetric key is bound to the principals holding the inverse key. While it does not make sense to bind the private key, an important goal in any public key system is that a public key is bound to the principal holding the private key. We explain this in Section 5.7.3, where we also give inference rules to meet the key binding requirements.

[1] An exception is when the key is meant to be forwarded, but in that case, it is usually opaque to A.

5.5 Actions

This section defines actions for protocols that use asymmetric keys. The actions are similar to those described in Chapter 4. They are performed by a principal, P_i during the run of a protocol. (The original eleven action appear in the complete list in Appendix A.)

12. Generate-key-pair(k^+, k^-)

result: $POSS(P_i) := POSS(P_i) \cup \{k^+ \bowtie P_i, k^- \bowtie P_i\}$

description: This action is used to generate keys that are the inverses of each other.

13. Apply-asymkey(X, k)

condition: $X, k \in POSS(P_i)$, k is an asymmetric key.

result: **if** $X = \{Y\}_{k'}$ **and** k is the inverse of k', **then** $POSS(P_i) := POSS(P_i) \cup Y$ **else** $POSS(P_i) := POSS(P_i) \cup \{\{X\}_k\}$

description: States that if an asymmetric key operation is performed on X, then there are two cases. Either X is of the form $\{Y\}_{k'}$ and k and k' are inverses, in which case the two keys cancel each other, or X is encrypted under k.

The *Asmkey-function* action is the analog of the *encrypt* and *decrypt* actions for symmetric keys. The binding of the keys, $k^+ \bowtie P_i$ and $k^- \bowtie P_i$, is explained in Section 5.7.3.

A new local set that contains the bindings generated by a principal, P_i, is defined.

Bindings Set(P_i) This set contains the legal bindings of keys held by a principal. These are bindings that are created by P_i, and bindings that are received in certificates from trusted servers. $Bindings(P_i) = \{k_1 \bowtie P_1, k_2 \bowtie P_2, \cdots, k_n \bowtie P_n\}$.

This set is needed because a principal must be able to identify bindings that it generated versus illegal bindings that it receives in messages. This is shown in Section 5.7.3.

A new action, which binds a principal to a key, is explained in Section 5.7.3 as well. The action is the same for symmetric and asymmetric keys.

14. Bind(k, P_j)

condition: $k \in POSS(P_i)$, k is a key intended for P_j.

result: $POSS(P_i) := POSS(P_i) \cup \{(k \bowtie P_j)\}$, $Bindings(P_i) := Bindings(P_i) \cup \{(k \bowtie P_j)\}$

description: States that if P_i possesses k, then after binding it to P_j, P_i possesses $k \bowtie P_j$. It is used to bind a key to a principal before sending it. $k \bowtie P_j$ is added to the *Bindings* set of P_i.

The binding of keys requires a change to the *Generate-secret* action. When a principal generates a secret, it binds that secret to itself. It is not necessary to update the *Bindings* set because this set is needed to maintain bindings to *other* principals.

4. Generate-secret(s)

result: $S := S \cup \{s\}$, $Observers(s) = \{P_i\}$, $POSS(P_i) := POSS(P_i) \cup \{s \bowtie P_i\}$,
$BEL(P_i) := BEL(P_i) \cup \#(s)$,

description: This action is used when a principal generates a secret data item, such as a key. A new secret, s, is added to S, and the *Observers* and possession sets are updated.

5.6 The Update function

The `Update` function is used to maintain the *Observers* sets after a message is sent. We repeat the algorithm here.

5.6.1 Notation

The cleartext messages in a message are numbered m_i. For example, in the message

$$\{x_1, \{x_2, k_3\}_{k_1}, x_4, x_2\}_{k_2}$$

$m_1 = x_1$, $m_2 = x_2$, $m_3 = k_3$, $m_4 = x_4$, and $m_5 = x_2$. $keys[m_i]$ is a set containing the keys appearing in m_i. In our example, $keys[m_3] = \{k_3\}$, and $keys[m_1] = \phi$. Finally, $needs[m_i]$ is the set of keys needed to see m_i. In our example, $needs[m_2] = \{k_1, k_2\}$, and $needs[m_4] = \{k_2\}$. These sets can be obtained by reading the keys used to encrypt a plaintext submessage from the inside out.

5.6.2 The Algorithm

When a message appears, each principal, P, inspects it, decrypts it insofar as possible, and accumulates any secrets found within.[2]

$$\forall \text{ interior } m_i \tag{5.1}$$

$$opaque(P) \mathrel{+}= m_i \tag{5.2}$$

$$\text{While changes occur to } haskeys(P) \tag{5.3}$$

$$\forall m_i \in opaque(P) \tag{5.4}$$

$$\text{if } needs[m_i] \subseteq haskeys(P) \tag{5.5}$$

$$opaque(P) \Leftrightarrow= m_i \tag{5.6}$$

$$seen(P) \mathrel{+}= m_i \tag{5.7}$$

$$haskeys(P) \mathrel{+}= keys[m_i] \tag{5.8}$$

5.6.3 The Extension

We extend this algorithm to work with asymmetric keys. If $\{x_i\}_{k^+}$ is a message, then $needs[m_i]$, the keys needed to read x_i, is equal to $\{k^-\}$. In general, when reading keys from the inside out to add to the *needs* sets, an occurrence of k^+ causes the corresponding k^- to included in the *needs* set, and an occurrence of k^- causes the corresponding k^+ to be included in the *needs* set. To illustrate, take:

$$\{x_1, \{x_2, k_3\}_{k_1{}^+}, x_4, x_2\}_{k_2}$$

In this example, $needs[m_2] = \{k_1{}^-, k_2\}$.

---

[2] The notation we use in this algorithm is similar to that of the C programming language. $X \mathrel{+}= Y$ is the same as $X := X + Y$.

5.7 Inference Rules

Our technique for protocol specification and analysis uses inference rules to reason about beliefs during the run of a protocol. These rules are applied whenever they are relevant. Here we extend the inference rules for asymmetric keys. (See Appendix C for a complete list of inference rules.)

5.7.1 The Linkage Rule

The rule dealing with freshness of nonces requires that the key needed to decrypt a message is fresh. Recall the linkage rule:

$$\frac{\begin{array}{c}\#(k) \in BEL(P), k^- \in POSS(P), \\ LINK(N_a) \in BEL(P), X \text{ contains } f(N_a), \\ X \text{ contains } x_1, \{X\}_k \text{ from } Q \in POSS(P)\end{array}}{BEL(P) := (BEL(P) \Leftrightarrow LINK(N_a)) \cup \{\#(x_1)\}}$$

This rule states that the submessages of a message X are believed to be fresh under certain conditions. If LINK(N_a) is in P's belief set, then the nonce N_a has not been used before. This is the first condition. If the rule is applied successfully, the LINK item is removed. So the rule could not fire again for the same nonce. Other conditions state that the nonce N_a must be sealed under a key that is fresh, and must be available to P.

This rule is slightly modified for asymmetric keys. In this case, the principal, P, must possess a fresh inverse key of the key that encrypts the message. There are two asymmetric key linkage rules. One rule is as follows:

$$\frac{\begin{array}{c}\#(k^-) \in BEL(P), k^- \in POSS(P), \\ LINK(N_a) \in BEL(P), X \text{ contains } f(N_a), \\ X \text{ contains } x_1, \{X\}_{k^+} \text{ from } Q \in POSS(P)\end{array}}{BEL(P) := (BEL(P) \Leftrightarrow LINK(N_a)) \cup \{\#(x_1)\}}$$

The other asymmetric key linkage rule is the same, except that the plus and minus signs on the key, k, are reversed.

The asymmetric key linkage rule above states that the submessages of a message X are believed to be fresh under certain conditions. If LINK(N_a) is in P's belief set, then the nonce N_a has not been used before. This is the first condition. If the rule is applied successfully, the LINK item is removed, and the rule does not fire

again for the same nonce. Other conditions state that the nonce N_a is sealed under k^+, and that the corresponding inverse key, k^- is believed to be fresh and available to P.

5.7.2 The Origin of Messages

In symmetric key systems, the origin of a message is usually inferred from the secret key that is used to encrypt it. A key is shared between two principals; as long as no other principals know it, this fact uniquely identifies the origin of a message.

In asymmetric key systems, this is not the case. Anyone in possession of a key can encrypt something that is readable only to principals in possession of the inverse key. In useful applications, at least one of the keys in the pair is available to more than two principals. When asymmetric keys are used as public/private keys, we assume that everyone is in possession of the public keys. Thus, to determine the origin of messages that are encrypted under public keys, we must look at the content of those messages. We introduce some new inference rules to deal with this.

We adopt the convention that an item is tagged as coming from any principal that can observe it. The reason is that logically, any principal that can observe X, can send a message containing X. The *Possible origins rule* handles this:

$$\frac{X \in POSS(P), X \text{ contains } x_1, R \in Obs(x_1), R \neq P}{x_1 \text{ from } R \in POSS(P)}$$

This rule states that submessages of X are marked as coming from all principals that observe them. For example, if $Observers(x_1)$ contains three elements (other than P), then three items are added to $POSS(P)$ by this rule.

The next rule deals with submessages of messages encrypted under asymmetric keys. It has a companion rule that is identical except that the minus and plus signs on the keys are reversed. The *Submessage origin rule* follows:

$$\frac{\begin{array}{c}\{X\}_{k^+} \in POSS(P), X \text{ contains } x_1 \text{ from } Q, \\ R \in Observers(k^-), X \text{ contains } x_2, R \neq P\end{array}}{x_2 \text{ from } Q \in POSS(P), x_2 \text{ from } R \in POSS(P)}$$

This rule states that if P possesses a message X encrypted under an asymmetric key, and if some submessage of X is marked as being from principal Q, then all

submessages of X are also marked as being from Q. In addition, the submessages are marked as being from any principal, other than P, who can observe the inverse key. This general rule can be simplified when public/private key systems are used, where we assume that each principal is in sole possession of its private key.

The simplification gives us two rules. The first one is used when a principal receives a messages encrypted under its public key. Note that a message encrypted under someone else's public key is opaque. The second one is used when a principal receives a message that is encrypted under someone else's private key. Also, it does not make sense for a principal to receive something encrypted under its own private key unless it was generated by the principal. The first rule is:

$$\frac{\{X\}_{k_p+} \in POSS(P), \quad X \text{ contains } x_1 \text{ from } Q, X \text{ contains } x_2}{x_2 \text{ from } Q \in POSS(P)}$$

This simplified version of the *Submessage origin rule* states that if P possesses a message, X, that is encrypted under its public key, then if X contains a submessage that is marked as being from a principal, Q, then all submessages of X are also marked as being from Q. This is true because the message could not have been tampered with by anyone other than the holder of the private key, in this case, P.

The second rule applies when a principal receives a message, X, encrypted under another principal, Q's, private key is:

$$\frac{\{X\}_{k_q-} \in POSS(P), X \text{ contains } x_2}{x_2 \text{ from } Q \in POSS(P)}$$

In this rule, the only conclusion we can reach, is that any submessage of X is from Q, because Q constructed the message.

The following example demonstrates the use of the *Submessage origin rule*. For simplicity, assume we are using a public/private key system with all the implied assumptions.[3] Say that Observers$(N_b) = \{A, B\}$, and that $N_b \in POSS(B)$. Then, by the *Possible origins rule*, N_b from $A \in POSS(B)$. Now, say that we have the following protocol step:

$$A \rightarrow B : \{K, N_b\}_{K_b+}$$

The *Submessage origin rule* can be instantiated with $X = \{K, N_b\}$.

[3]Each principal is in sole possession of its private key and public keys are available to all.

$$\frac{\{X\}_{k^+} \in POSS(B), X \text{ contains } N_b \text{ from } A, \quad R \in Observers(k^-), X \text{ contains } K, R \neq B}{K \text{ from } A \in POSS(B), K \text{ from } R \in POSS(B)}$$

The first condition states that $\{X\}_{k^+} \in POSS(B)$. This is true after the message is received. The second condition is X contains N_b from A. This condition holds as shown above. The next condition states that $R \in Observers(k^-)$. The purpose of this condition is to bind R. Because $R \neq P$, and $Observers(k^-) = \{P\}$, we conclude that $R = \phi$. The final condition states that X contains K. The result of the rule is K from $A \in POSS(B)$ and K from $R \in POSS(B)$. The second part can be ignored because R is null.

This demonstrates how B can obtain the knowledge that the key, K, comes from A. If there is no other *from* tag associated with K in B's possession, then B can confidently assume that K is indeed from A, and not a replay. Note that the conclusion is entirely based on the earlier statement that Observers$(N_b) = \{A, B\}$.

This example shows how a nonce can be used to obtain confidence in the origin of a message. We have said nothing about freshness here, but it follows that if N_b is a fresh nonce, then K is a fresh key. In Section 5.8.1 we use this example in a sample protocol to demonstrate protocol analysis.

5.7.3 Binding Keys to Principals

In section 5.4 we introduced the notion of binding a key to a principal. We extend our technique for analyzing protocols, and require that each symmetric session key be bound to the principal receiving the binding.

For asymmetric keys, we require that a key be bound to all principals in the observers set of the inverse key. In public/private key systems, the public key is bound to the holder of the private key, and we do not require the private key to be bound.

The *Bindings* sets play an important role in the analysis of protocols. Every principal maintains a set of the bindings that it generated. If a principal receives a symmetric key that is bound to some principal other than itself, then the protocol must abort. Similarly, an asymmetric key that is received in a message, and is not bound to an inverse key in the possession of the principal, results in an abort. The

Bindings sets enable a principal to distinguish key bindings it possesses that were received in messages, and ones that it generated itself.

The *Bindings* set contains all bindings that a principal possesses that should not cause the protocol to abort. Bindings that are received in certificates from trusted servers are included in this set. The *Certificate binding rule* takes precedence over the other rules introduced in this section. This rule adds a binding from a trusted server to the *Bindings* set of a principal.

$$\frac{(k \bowtie Q) \text{ from } S \in POSS(P), Trusted(S) \in BEL(P)}{Bindings(P) := Bindings(P) \cup \{(k \bowtie Q)\}}$$

The statement $Trusted(S)$ means that S is a trusted server, and it is specified in the initial assumptions of a protocol.

We now give inference rules to enforce the requirements mentioned above. The *Unbound key rule* makes a key that is not bound, observable by anyone. Recall that $\mathcal{W}$ is a special symbol representing all principals. This rule holds for symmetric keys and public keys. Private keys are not included.

$$\frac{k \in POSS(P),\ \nexists Q : (k \bowtie Q) \in POSS(P)}{Observers(k) := \mathcal{W}}$$

This rule states that if P possesses k, and there is no principal that k is bound to, then all principals can observe k. That is, k is no longer considered a secret key. Note that for this rule to apply, it is sufficient for one principal to possess an unbound copy of a key.

We introduce the *Bound key origin rule* for symmetric keys.

$$\frac{(k \bowtie Q) \in POSS(P), Q \neq P,\ (k \bowtie Q) \notin Bindings(P)}{Abort}$$

This rule states that if P possesses the key, k, that is bound to principal Q, and $Q \neq P$, then if this binding is not in the set of legal bindings for P, the protocol should abort. The set of legal bindings includes bindings created by P, and bindings received in certificates from trusted servers.

The *Bound key origin rules* for asymmetric keys are:

$$\frac{(k^+ \bowtie Q) \in POSS(P), (k^+ \bowtie Q) \notin Bindings(P),\ \{X\}_{k^-} \text{ from } R \in POSS(P), R \neq Q}{Abort}$$

and

$$\frac{(k^- \bowtie Q) \in POSS(P), (k^- \bowtie Q) \notin Bindings(P), \quad \{X\}_{k^+} \text{ from } R \in POSS(P), R \neq Q}{Abort}$$

These rules state that if P possesses an asymmetric key bound to Q, and P possesses a message that is encrypted under the inverse key, then the message must come from Q. Otherwise, the protocol aborts. Of course, only the first of these rules applies in the case of public/private key systems.

It is easy to see that when our analysis is applied to the Denning and Sacco protocol [17] described in Section 5.4, after the third step, $Observers(k) = \mathcal{W}$. Thus, although the details are not provided here, it is clear that our analysis uncovers the flaw.

5.8 Examples

We use two examples to demonstrate the specification and analysis technique for asymmetric keys. Our first example is a simple public key protocol where one principal generates a symmetric key to be used with another principal. The second example is the Needham and Schroeder public key protocol [48]. We present the protocol, analyze it to uncover the known flaw, and then present a fix. Finally, we analyze the fix using our method.

5.8.1 A Sample Protocol

In this section, we present a protocol that demonstrates the extended technique for specifying and analyzing cryptographic protocols that use asymmetric keys. We have two principals, A and B, and we assume that each has a fresh copy of the other's public key. We also assume that each principal is the only holder of its own private key. A generates a symmetric key, and sends it to B. The goal is that A and B are the exclusive holders and observers of the key. Roughly, the protocol is as follows:

1. $A \rightarrow B : N_a$
2. $B \rightarrow A : \{N_b\}_{k_a{}^+}$

3. $A \rightarrow B : \{k_{ab}, A, N_b\}_{K_b{}^+}$

4. $B \rightarrow A : \{N_a\}_{k_{ab}}$

In an earlier version, we omitted the encryption of message 2, and were not able to obtain the desired goal. Analysis revealed the obvious flaw, that anyone could have generated message 3. The encryption of message 2 with A's public key fixes this, and we are able to reach the desired goals. Our analysis does not reveal any further flaws, and we believe the protocol to be correct.

5.8.1.1 The Specification

The specification of the protocol is as follows:

Principal A

$POSS(A) = \{k_a{}^-, k_a{}^+ \bowtie A, k_b{}^+ \bowtie B\}$
$BEL(A) = \{\#(k_a{}^-), \#(k_a{}^+), \#(k_b{}^+)\}$
$BL(A) =$

- Generate-nonce(N_a)
  Send($B, \{N_a\}$)
  Update($\{N_a\}$)
  Receive($B, \{N_b\}_{k_a{}^+}$)
  Apply-asymkey($\{N_b\}_{k_a{}^+}, k_a{}^-$)
  Generate-secret(k_{ab})
  Bind(k_{ab}, A)
  Send(B,Apply-asymkey(Concat($k_{ab} \bowtie A$,
  N_b)), $K_b{}^+$)
  Update($\{k_{ab} \bowtie A \cdot N_b\}_{K_b{}^+}$)
  Receive($B, \{N_a\}_{k_{ab}}$)
  Decrypt($\{N_a\}_{k_{ab}}, k_{ab}$)

Principal B

$POSS(B) = \{k_b{}^-, k_a{}^+ \bowtie A, k_b{}^+ \bowtie B\}$
$BEL(B) = \{\#(k_b{}^-), \#(k_a{}^+), \#(k_b{}^+)\}$

$BL(A) =$

Receive($A, \{N_a\}$)

Generate-nonce(N_b)

Send(A,Apply-asymkey($\{N_b\}, {k_a}^+$))

Update($\{N_b\}_{{k_a}^+}$)

Receive($A, \{k_{ab} \bowtie A \cdot N_b\}_{{K_b}^+}$)

Apply-asymkey($\{k_{ab} \bowtie A \cdot N_b\}_{{K_b}^+}, {k_b}^-$)

Split($\{k_{ab} \bowtie A \cdot N_b\}$)

Send(A,Encrypt($\{N_a\}, k_{ab}$))

Update($\{N_a\}_{k_{ab}}$)

5.8.1.2 The Analysis

In this section we discuss the analysis of the protocol. We will focus on the highlights of the protocol and omit the tedious details.

After the first `Update` action by principal B,

- Update($\{N_b\}_{{k_a}^+}$)

Observers(N_b) = {A,B}. The first `Apply-asymkey` in principal A's behavior list is the first interesting point in the analysis.

- Apply-asymkey($\{N_b\}_{{k_a}^+}, {k_a}^-$)

After this action, N_b is added to $POSS(A)$. Now, the *Possible origins rule* applies. Because Observers(N_b) = {A,B}, and thus, no other principals can observe N_b, B is the only principal that satisfies the conditions for R in the rule. Therefore, N_b from B is added to $POSS(A)$. In section 5.7.2 we showed that given N_b from $B \in POSS(A)$, the analysis leads to the conclusion that the only possible occurrence of the symmetric key with a *from* label in $POSS(B)$ is k_{ab} from A. This is consistent with the binding of k_{ab} to A, so the *Bound key origin rule* does not apply, and the protocol is not aborted.

Applying the inference rules for message linking and freshness, we also conclude that k_{ab} is a fresh key because it is linked with the nonce N_b. After the final `Receive` action by principal A,

1. $A \rightarrow S : A, B$
2. $S \rightarrow A : \{K_b{}^+, B\}_{K_s{}^-}$
3. $A \rightarrow B : \{N_a, A\}_{K_b{}^+}$
4. $B \rightarrow S : B, A$
5. $S \rightarrow B : \{K_a{}^+, A\}_{K_s{}^-}$
6. $B \rightarrow A : \{N_a, N_b\}_{K_a{}^+}$
7. $A \rightarrow B : \{N_b\}_{K_b{}^+}$

Figure 5.1: **The Needham and Schroeder public key protocol specification.** Protocols are specified by a principal name, followed by an arrow and another principal name, followed by a message.

- Receive$(B, \{N_a\}_{k_{ab}})$

the linkage rule is applied, and we conclude that A and B now share a secret key that is fresh.

5.8.2 Needham and Schroeder

The Needham and Schroeder public key protocol [48], as it is originally specified, is given in Figure 5.1. Burrows *et al.* [9] point out a weakness in this protocol. We show how this weakness is found using our analysis technique. Then, we provide a fix, and show that the weakness no longer exists.

5.8.2.1 The Specification

We now give the specification of the protocol. Again, we assume a public/private key system; however, we do not assume that the public keys are available to everyone. They must be received in a certificate from a trusted server.

Principal A

$POSS(A) = \{k_a{}^-, k_a{}^+ \bowtie A, k_s{}^+ \bowtie S\}$

$BEL(A) = \{\#({k_a}^-), \#({k_a}^+), \#({k_s}^+), Trusted(S)\}$

$BL(A) =$

- Send(S, {Concat(A, B)})
  Update({A,B})
  Receive($S, \{{k_b}^+ \bowtie B\}_{K_s-}$)
  Apply-asymkey($\{{k_b}^+ \bowtie B\}_{K_s-}, {K_s}^+$)
  Generate-nonce(N_a)
  Send(B,Apply-asymkey(Concat(N_a, A),${k_b}^+$)
  Update($\{N_a, A\}_{{k_b}^+}$)
  Receive($B, \{N_a \cdot N_b\}_{{k_a}^+}$)
  Apply-asymkey($\{N_a \cdot N_b\}_{{k_a}^+}, {k_a}^-$)
  Send(B,Apply-asymkey($\{N_b\}, {k_b}^+$))
  Update($\{N_b\}_{{k_b}^+}$)

Principal B

$POSS(B) = \{{k_b}^-, {k_b}^+ \bowtie B, {k_s}^+ \bowtie S\}$

$BEL(B) = \{\#({k_b}^-), \#({k_b}^+), \#({k_s}^+), Trusted(S)\}$

$BL(B) =$

  Receive($A, \{N_a, A\}_{{k_b}^+}$)
  Apply-asymkey($\{N_a, A\}_{{k_b}^+}, {k_b}^-$)
  Send(S,Concat(B, A))
  Update($\{B \cdot A\}$)
  Receive($S, \{{k_a}^+ \bowtie A\}_{K_s-}$)
  Apply-asymkey($\{{k_a}^+ \bowtie A\}_{K_s-}, {K_s}^+$)
  Generate-nonce(N_b)
  Send(A, Apply-asymkey(Concat(N_a, N_b),${k_a}^+$))
  Update($\{N_a \cdot N_b\}_{{k_a}^+}$)
  Receive($A, \{N_b\}_{{k_b}^+}$)
  Apply-asymkey($\{N_b\}_{{k_b}^+}, {k_b}^-$)

Principal S

$POSS(S) = \{{k_s}^-, {k_s}^+ \bowtie S, {k_a}^+ \bowtie A, {k_b}^+ \bowtie B\}$
$BEL(S) = \{\#({k_s}^-, \#({k_s}^+), \#({k_a}^+), \#({k_b}^+)\}$
$BL(S) =$
Receive$(A, \{A \cdot B\})$
Send(A,Apply-asymkey$(\{{k_b}^+ \bowtie B\}, {K_s}^-)$
Update$(\{{k_b}^+ \bowtie B\}_{K_s}-)$
Receive$(B, \{B \cdot A\})$
Send(B,Apply-asymkey$(\{{k_a}^+ \bowtie A\}, {K_s}^-))$
Update$(\{{k_a}^+ \bowtie A\}_{K_s}-)$

Note that the certificates from the trusted server, S, are of the form $\{{k_b}^+ \bowtie B\}$. Thus, ${k_b}^+$ is bound to B. The *Certificate binding rule* applies and this binding is added to *Bindings(A)*, and the *Bound key origin rules* does not apply.

5.8.2.2 The Analysis

In this section, we discuss the analysis of the protocol. Again we focus on the interesting aspects of the analysis and omit the details that are not necessary for this demonstration.

The third action in A's behavior list is:

Receive$(S, \{{k_b}^+ \bowtie B\}_{K_s}-)$

This is decrypted in the next action,

Apply-asymkey$(\{{k_b}^+ \bowtie B\}_{K_s}-, {K_s}^+)$

The result is that ${k_b}^+ \bowtie B$ from S is added to $POSS(A)$. Because the message from S is not linked with a nonce or a time stamp, there is no way to conclude that ${k_b}^+$ is fresh. Our technique requires that a key be fresh for anything encrypted under that key to be fresh. Thus, after A sends $\{N_a, A\}_{{k_b}^+}$, B cannot conclude that N_a is fresh.

Similarly, B cannot conclude that ${k_a}^+$ is fresh. Thus, when B sends $\{N_a \cdot N_b\}_{{k_a}^+}$, A cannot conclude that N_b is fresh. Thus, B does not conclude that N_b is fresh. In fact, nothing encrypted under ${k_a}^+$ or ${k_b}^+$ is fresh, and the protocol fails

to establish a secure channel for fresh communication because the public keys being used are not fresh.

5.8.2.3 The Fix

The problem with the Needham and Schroeder Public-key protocol is that the certificates for the public keys of A and B are not linked to the requests. A simple modification takes care of this. Each principal includes a nonce with the request for a certificate, and this nonce is included with the certificate. This will result in the conclusion that the public keys are fresh, and thus, anything encrypted under them is fresh. The fixed protocol follows:

Principal A

$POSS(A) = \{{k_a}^-, {k_a}^+, {k_s}^+\}$
$BEL(A) = \{\#({k_a}^-), \#({k_a}^+), \#({k_s}^+)\}$
$BL(A) =$

- Generate-nonce($N_a{}'$)
  Send(S, {Concat($A, B, N_a{}'$)})
  Update($\{A \cdot B \cdot N_a{}'\}$)
  Receive($S, \{{k_b}^+ \bowtie B \cdot N_a{}'\}_{K_s-}$)
  Apply-asymkey($\{{k_b}^+ \bowtie B \cdot N_a{}'\}_{K_s-}, {K_s}^+$)
  Generate-nonce(N_a)
  Send(B,Apply-asymkey(Concat(N_a, A),${k_b}^+$)
  Update($\{N_a, A\}_{{k_b}^+}$)
  Receive($B, \{N_a \cdot N_b\}_{{k_a}^+}$)
  Apply-asymkey($\{N_a \cdot N_b\}_{{k_a}^+}, {k_a}^-$)
  Send(B,Apply-asymkey($\{N_b\}, {k_b}^+$))
  Update($\{N_b\}_{{k_b}^+}$)

Principal B

$POSS(B) = \{{k_b}^-, {k_b}^+, {k_s}^+\}$
$BEL(B) = \{\#({k_b}^-), \#({k_b}^+), \#({k_s}^+)\}$
$BL(B) =$

Receive($A, \{N_a, A\}_{k_b{}^+}$)

Apply-asymkey($\{N_a, A\}_{k_b{}^+}, k_b{}^-$)

Generate-nonce($N_b{}'$)

Send(S,Concat($B, A, N_b{}'$))

Update($\{B \cdot A \cdot N_b{}'\}$)

Receive($S, \{k_a{}^+ \bowtie A \cdot N_b{}'\}_{K_s{}^-}$)

Apply-asymkey($\{k_a{}^+ \bowtie A \cdot N_b{}'\}_{K_s{}^-}, K_s{}^+$)

Generate-nonce(N_b)

Send(A, Apply-asymkey(Concat(N_a, N_b),$k_a{}^+$))

Update($\{N_a \cdot N_b\}_{k_a{}^+}$)

Receive($A, \{N_b\}_{k_b{}^+}$)

Apply-asymkey($\{N_b\}_{k_b{}^+}, k_b{}^-$)

Principal S

$POSS(S) = \{k_s{}^-, k_s{}^+, k_a{}^+, k_b{}^+\}$

$BEL(S) = \{\#(k_s{}^-, \#(k_s{}^+), \#(k_a{}^+), \#(k_b{}^+)\}$

$BL(S) =$

Receive($A, \{A \cdot B \cdot N_a{}'\}$)

Send(A,Apply-asymkey(Concat($k_b{}^+ \bowtie B, N_a{}'$), $K_s{}^-$)

Update($\{k_b{}^+ \bowtie B \cdot N_a{}'\}_{K_s{}^-}$)

Receive($B, \{B \cdot A \cdot N_b{}'\}$)

Send(B,Apply-asymkey(Concat($k_a{}^+ \bowtie A, N_b{}'$), $K_s{}^-$))

Update($\{k_a{}^+ \bowtie A \cdot N_b{}'\}_{K_s{}^-}$)

5.8.2.4 Analysis of revised protocol

The messages from S that contain public keys, $\{k_b{}^+ \bowtie B\}_{K_s{}^-}$ and $\{k_a{}^+ \bowtie A\}_{K_s{}^-}$ are certificates. They represent the assertion by S that $k_b{}^+$ is bound to B and $k_a{}^+$ is bound to A.

The difference between the original protocol and the revised version is that the certificates for the public keys are linked to the individual requests. Principal A

sends the nonce $N_a{}'$ to S in the clear. When the nonce is generated, $LINK(N_a{}')$ is added to $BEL(A)$, to guarantee that the nonce is only used once. As the returned nonce is sealed under the public key of S, nobody can tamper with it. This is guaranteed in the protocol by the *Possible origins rule*. Because $Observers(k_s{}^-) = S$, A concludes that the message is from S. Then, the *Submessage origin rule* and the *Linkage rule* are used to conclude that the public key for B is fresh.

Given that $\#(k_b{}^+) \in BEL(A)$ and $\#(k_a{}^+) \in BEL(B)$, it follows that the remaining messages linked with nonces contain fresh messages. The analysis concludes that the final message from A to B, $\{N_b\}_{k_b{}^+}$ is fresh.

5.9 Conclusions

We extend the specification and analysis technique of Chapter 4 for protocols that use asymmetric keys, provide a sample protocol to illustrate the extended technique.

We introduce the notion of binding a key to a principal, and show why this is important for detecting flaws in protocols. We use the example by Abadi and Needham [1] to demonstrate.

Our analysis reveals the weakness in the Needham and Schroeder public key protocol [48], that was discovered by Burrows *et al.* [9] We provide a fix to this protocol that eliminates the weakness. Finally, we analyze the revised protocol and show that the weakness no longer exists.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

This thesis makes several contributions:

1. A solution, *khat*, is provided to the problem of long-running jobs in an authenticated environment. This service has been implemented in Kerberos and is compatible with AFS [28].

2. A new specification and analysis technique is introduced. The protocol designer specifies the actions of each principal. These actions are also used to model the knowledge of the principals. The beliefs of principals are propagated via inference rules similar to those used in the BAN logic [9]. In addition, each message sent on the network is considered a broadcast, and *Observers* sets are maintained for each secret to represent what each principal can learn by examining network traffic. The new technique contains several advantages over existing methods.

   (a) Protocols that are not monotonic, because they rely on the deletion of information, can be specified. *Khat* is an example of such a protocol.

   (b) The specification is designed to closely model the underlying distributed system, and thus, can be easily converted to an implementation.

   (c) A protocol can be tested to see if it is susceptible to known attacks by modeling the actions of the intruder.

   (d) The assumption made by the protocol designer are specified explicitly. This is valuable because flaws are often introduced into protocols as a result of confusion from unclear assumptions.

(e) The use of nonces is restricted to one challenge/response pair. This avoids flaws in protocols that stem from improper use of nonces. This restriction helps the protocols designer avoid replay attacks that result from naive impelmentation of nonces.

(f) Flaws that result from unbound keys can be detected. The technique presented in this thesis requires that all keys be bound to some principal, and illegal bindings are detected. This helps the protocol designer detect certain flaws that permit an intruder to masquerade as another principal.

3. The observation is made that by eliminating the freeness of variables, such as nonces and keys, protocols can be developed with higher assurance.

In light of this work, there are new problems to explore. It is an open problem whether or not a logic can be developed for nonmonotonic protocols that is *sound* and *complete*. A logic is sound if it is impossible to derive a contradiction. It is complete if anything that is true can be derived. These two properties are required in order to prove that no flaws exist in a protocol [66]. The only known semantics for a logic of authentication, the Abadi Tuttle semantics [2], was unsuccessful in proving soundness and completeness.

Other projects for future work include:

- Verifiable plaintext:
  Exhaustive search on a key space is possible when the plaintext in a message is recognizable. The decryption algorithm is applied using all possible keys, and when the recognized plaintext is revealed, the key is found. In many cases, it is possible to eliminate this risk. In the future, we hope to add the ability to reason about verifiable plaintext to our technique.

- Automated tool:
  Many of the steps in the specification and analysis technique can be automated. In the future, we hope to develop an interactive tool for specifying and analyzing protocols. This will improve the usefulness of our technique, by making it available to more people.

- Analyzing real-world protocols:
  In the future, we hope to use the technique presented here to analyze more

protocols that are used in practice. For example, standards are being developed for privacy enhanced mail (PEM) [3], and it would be valuable to explore the security of the proposed schemes.

BIBLIOGRAPHY

[1] M. Abadi and R. Needham. Good engineering practice for authentication protocol design. *Manuscript*, 1993.

[2] Martin Abadi and Mark R. Tuttle. A semantics for a logic of authentication. *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, pages 201–216, August 1991.

[3] D. Balenson. Privacy enhancement for Internet electronic mail: part iii—algorithms, modes, and identifiers. *RFC 1423*, February 1993.

[4] P. Bieber. A logic of communication in a hostile environment. *Proceedings of the Computer Security Foundation Workshop III*, pages 14–22, June 1990.

[5] Thomas Blumer and Deepinder P. Sidhu. Mechanical verification and automatic implementation of communication protocols. *IEEE Transactions on Software Engineering*, SE-12(8):827–843, August 1986.

[6] D.E. Britton. Formal verification of a secure network with end-to-end encryption. *Proceedings of the 1984 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 154–166, May 1984.

[7] L. Brownston and E. Kant. *Programming Expert Systems in OPS5.* Addison Wesley, 1985.

[8] J. Burns and C. J. Mitchell. A security scheme for resource sharing over a network. *Computers and Security*, 9:67–76, February 1990.

[9] M. Burrows, M. Abadi, and R. Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8, February 1990.

[10] M. Burrows, M. Abadi, and R. Needham. Rejoinder to Nessett. *Operating System Review*, 24(2):39–40, April 1990.

[11] Claudio Calvelli and Vijay Varadharajan. An analysis of some delegation protocols for distributed systems. *Proceedings of the Computer Security Foundation Workshop V*, pages 92–110, 1992.

[12] E. A. Campbell, R. Safavi-Naini, and P. A. Pleasants. Partial belief and probabilistic reasoning in the analysis of secure protocols. In *Proceedings of the Computer Security Foundation Workshop V*, pages 84–91, Washington, 1992.

[13] B. F. Chellas. *Modal Logic: An Introduction.* Cambridge University Press, 1980.

[14] P. C. Cheng and V.D. Gligor. On the formal specification and verification of a multiparty session protocol. *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 216–233, May 1990.

[15] W. F. Clocksin and C. S. Mellish. *Programming in Prolog.* Springer-Verlag, 1984.

[16] D. W. Davies and W.L. Price. *Security for Computer Networks.* Wiley, 1984.

[17] Dorothy E. Denning and Giovanni Maria Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, August 1981.

[18] W. Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6), 1976.

[19] D. Dolev and A. Yao. On the security of public-key protocols. *Communications of the ACM*, 29:198–208, August 1983.

[20] M. Fitting. *Proof Methods for Modal and Intuitionistic Logics.* D. Reidel Publishing Company, 1983.

[21] Klaus Gaarder and Einar Snekkenes. Applying a formal analysis technique to the CCITT X.509 strong two-way authentication protocol. *Journal of Cryptology*, 3:81–98, 1991.

[22] Li Gong. Handling infeasible specifications of cryptographic protocols. *Proceedings of the Computer Security Foundation Workshop IV*, pages 99–102, 1991.

[23] Li Gong, Roger Needham, and Raphael Yahalom. Reasoning about belief in cryptographic protocols. *Proceedings of the 1990 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 234–248, May 1990.

[24] J. Y. Halpern and M. Y. Vardi. The complexity of reasoning about knowledge and time. *Proceedings of the Eighteenth ACM Symposium on the Theory of Computing*, pages 304–415, 1986.

[25] J. Hintikka. *Knowledge and Belief.* Cornell University Press, 1962.

[26] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10):576–580, 1969.

[27] John E. Hopcroft and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[28] J.H. Howard, M.L. Kazar, S.G. Menees, D.A. Nichols, M. Satyanarayanan, R.N. Sidebotham, and M.J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[29] James W. Gray III and Paul F. Syverson. A logical approach to multilevel security of probabilistic systems. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 164–176, May 1992.

[30] D. Kahn. *The Codebreakers*. Macmillan Publishing Co., 1967.

[31] R. Kailar and V. D. Gilgor. On belief evolution in authentication protocols. *Proceedings of the Computer Security Foundation Workshop IV*, pages 103–116, June 1991.

[32] T. Kasami, S. Yamamura, and K. Mori. A key management scheme for end-to-end encryption and a formal verification of its security. *Systems, Computers, Control*, 13:59–69, May-June 1982.

[33] Richard A. Kemmerer. Analyzing encryption protocols using formal verification techniques. *IEEE Journal on Selected areas in Communications*, 7(4):448–457, May 1989.

[34] A. Kerckhoffs. *La Cryptographie Militaire*. Libraire Militaire de L. Baudoin & Cie, Paris, 1883.

[35] B. Lampson, M. Abadi, M. Burrows, and E. Wobber. Authentication in distributed systems: Theory and practice. *ACM Transactions on Computer Systems*, 10(4), November 1992.

[36] D. Longley and S. Rigby. Use of expert systems in the analysis of key management systems. *Security and Protection in Information Systems*, pages 213–224, 1989.

[37] W. P. Lu and M. K. Sundareshan. Secure communication in Internet environments: A hierarchical key management scheme for end-to-end encryption. *IEEE Transactions on Communications*, 37(10):1014–1023, October 1989.

[38] Wenbo Mao and Colin Boyd. Towards formal analysis of security protocols. *Proceedings of the Computer Security Foundation Workshop VI*, pages 147–158, June 1993.

[39] Catherine Meadows. Using narrowing in the analysis of key management protocols. *Proceedings of the 1989 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 138–147, May 1989.

[40] Catherine Meadows. Representing parital knowledge in an algebraic security model. *Proceedings of the Computer Security Foundation Workshop III*, pages 23–31, June 1990.

[41] Catherine Meadows. A system for the specification and analysis of key management protocols. *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 182–195, May 1991.

[42] Catherine Meadows. Applying formal methods to the analysis of a key management protocol. *Journal of Computer Security*, 1(1):5–35, 1992.

[43] Michael J. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, 1983.

[44] Jonathan K. Millen, Sidney C. Clark, and Sheryl B. Freedman. The interrogator: Protocol security analysis. *IEEE Transactions on Software Engineering*, SE-13(2):274–288, February 1987.

[45] Judy H. Moore. Protocol failures in cryptosystems. *Proceedings of the IEEE*, 76(5):594–602, May 1988.

[46] Louise E. Moser. A logic of knowledge and belief for reasoning about computer security. *Proceedings of the Computer Security Foundation Workshop II*, pages 57–63, 1989.

[47] National Bureau of Standards. Data encryption standard. *Federal Information Processing Standards Publication*, 1(46), 1977.

[48] R.M. Needham and M.D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[49] R.M. Needham and M.D. Schroeder. Authentication revisited. *Operating Systems Review*, 21:7, January 1987.

[50] D. M. Nessett. A critique of the Burrows, Abadi and Needham logic. *Operating System Review*, 24(2):35–38, April 1990.

[51] D. Otway and O. Rees. Efficient and timely mutual authentication. *ACM Operating System Review*, 21(1):8–10, January 1987.

[52] P. V. Rangan. An axiomatic basis of trust in distributed systems. *Proceedings of the 1988 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 204–211, May 1988.

[53] P. Rety, C. Kirchner, H. Kirchner, and P. Lescanne. Narrower: a new algorithm for unification and its application to logic programming. *Rewriting Techniques and Applications, Lecture Notes in Computer Science*, 202, 1985.

[54] A. D. Rubin and P. Honeyman. Long running jobs in an authenticated environment. *USENIX Security Conference IV*, pages 19–28, October 1993.

[55] J.H. Saltzer and M. D. Schroeder. The protection of information in computer systems. *Proc. of the IEEE*, 63(9), September 1975.

[56] M. Sato. Study of Kripke-style models of some modal logics by Gentzen's sequential method. *Publications of the Research Institute for Mathematical Sciences*, 13(2), 1977.

[57] J. Scheid and S. Holtsberg. *Ina Jo Specification Language Reference Manual.* Systems Development Group, Unisys Corporation, September 1988.

[58] Bruce Schneier. *Applied Cryptography - Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, Inc., 1994.

[59] Yoav Shoham and Yoram Moses. Belief as defeasible knowledge. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, pages 1168–1173, August 1989.

[60] Deepinder P. Sidhu. Authentication protocols for computer networks: I. *Computer Networks and ISDN Systems*, 11:297–310, 1986.

[61] Einar Snekkenes. Authentication in open systems. *Proceedings of the IFIP WG 6.1 Tenth International Symposium on Protocol Specification, Testing, and Verification*, pages 311–324, June 1990.

[62] Einar Snekkenes. Exploring the BAN approach to protocol analysis. *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 171–181, May 1991.

[63] Einar Snekkenes. Roles in cryptographic protocols. *Proceedings of the 1992 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 105–119, May 1992.

[64] J.G. Steiner, B.C. Neuman, and J.I. Schiller. Kerberos: An authentication service for open network systems. In *Usenix Conference Proceedings*, pages 191–202, Dallas, Texas, February 1988.

[65] Paul Syverson. A logic for the analysis of cryptographic protocols. Technical Report 9305, Naval Research Laboratory, December 19.

[66] Paul Syverson. Formal semantics for logics of cryptographic protocols. *Proceedings of the Computer Security Foundation Workshop III*, pages 32–41, June 1990.

[67] Paul Syverson. The use of logic in the analysis of cryptographic protocols. *Proceedings of the 1991 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 156–170, May 1991.

[68] Paul Syverson. Adding time to a logic of authentication. *1st ACM Conference on Computer and Communications Security*, pages 97–101, November 1993.

[69] Paul Syverson. On a key distribution protocol of Newman and Stubblebine. *Submitted to Operating System Review*, 1993.

[70] Paul Syverson and Catherine Meadows. A logical language for specifying cryptographic protocol requirements. *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 165–177, May 1993.

[71] Paul F. Syverson. Knowledge, belief, and semantics in the analysis of cryptographic protocols. *Journal of Computer Security*, 1:317–334, 1992.

[72] Bhavani Thurasingham. A nonmonotonic multilevel logic for multilevel secure data/knowledge base management systems - II. *Proceedings of the Computer Security Foundation Workshop V*, pages 135–146, June 1992.

[73] M. J. Toussaint. A new method for analyzing the security of cryptographic protocols. *Journal of Selected Areas in Communication*, 11(5):702–714, June 1993.

[74] G.W. Treese. Berkeley UNIX on 1000 workstations: Athena changes to 4.3 BSD. *USENIX Winter COnference, Dallas, Texas*, pages 175–182, February 1988.

[75] V. Varadharajan and S. Black. Formal specification of a secure distributed system. *Proceedings of the 12th National Computer Security Conference*, pages 146–171, October 1989.

[76] Vijay Varadharajan. Verification of network security protocols. *Computers and Security*, 8(8):693–708, 1989.

[77] Vijay Varadharajan. Use of a formal description technique in the specification of authentication protocols. *Computer Standards and Interfaces*, 9:203–215, 1990.

[78] V. L. Voydock and S. T. Kent. Security mechanisms in high–level network protocols. *Computing Surveys*, 15(2):135–171, June 1983.

[79] C. H. West. General technique for communications protocol validation. *IBM Journal of Research and Development*, 22:393–404, 1978.

[80] Thomas Y.C. Woo and Siman S. Lam. A semantic model for authentication protocols. *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 178–194, May 1993.

APPENDICES

APPENDIX A

ACTIONS

1. **Encrypt**(X, k)

   condition: $X, k \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup \{\{X\}_k\}$

   description: This action is used when a principal encrypts data. If P_i possesses X and knows k then he can possess $\{X\}_k$.

2. **Decrypt**$(\{X\}_k, k)$

   condition: $\{X\}_k, k \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup \{X\}$

   description: This action is used when a principal decrypts data. If P_i possesses X, encrypted under k, and P_i knows k, then P_i can possess X.

3. **Generate-nonce(N)**

   result: $POSS(P_i) := POSS(P_i) \cup \{N\}$,
   $BEL(P_i) := BEL(P_i) \cup LINK(N)$

   description: This action is used when a principal generates a nonce to link a challenge and a response. LINK(N) is removed from $BEL(P_i)$ when the response is received. This is used to determine freshness.

4. **Generate-secret(s)**

   result: $S := S \cup \{s\}$, $Observers(s) = \{P_i\}$, $POSS(P_i) := POSS(P_i) \cup \{s \bowtie P_i\}$,
   $BEL(P_i) := BEL(P_i) \cup \#(s)$

   description: This action is used when a principal generates a secret data item, such as a key. A new secret, s, is added to S, and the *Observers* and possession sets are updated.

5. **Concat**$(X_1, X_2, \cdots, X_n)$

   condition: $X_1, X_2, \cdots, X_n \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup$
   $\{X_1 \cdot X_2 \cdots X_n\}$

   description: This action is used when a principal constructs a message, X, out of submessages $X_1, X_2, \cdots, X_n$.

6. **Split**(X)

   condition: X *contains* $x_1 \cdot x_2 \cdots x_n$,
   $X \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup \{x_1, x_2, \cdots, x_n\}$

   description: This action is used to break a message into its components. Split is the opposite of concatenation.

7. **Forget**(X)

   condition: $X \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \Leftrightarrow \{X\}, \forall\, P_j \in P$ **if** $TRUST[j,i] = 1$ **then**
   $BEL(P_j) := BEL(P_j) \Leftrightarrow \{X \in POSS(P_i)\}$

   description: This action is used when P_i no longer is in possession of X. All principals who trust P_i believe that P_i no longer possesses X.

8. **Forget-secret**(s)

   condition: $s \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \Leftrightarrow \{s\}$,
   $\forall\, P_j \in P$ **if** $TRUST[j,i] = 1$ **then**
   $BEL(P_j) := BEL(P_j) \Leftrightarrow \{s \in POSS(P_i)\}$

   description: This action is used when P_i no longer knows the secret s. All principals who trust P_i believe that P_i no longer possesses s.

9. **Apply**(f, X)

   condition: $f, X \in POSS(P_i)$

   result: $POSS(P_i) := POSS(P_i) \cup \{f(X)\}$

description: This action is used when P_i applies the function f to X. After the application, P_i possesses $f(X)$.

10. Check-freshness(X)

condition: $X \in POSS(P_i)$, X has not expired

result: $BEL(P_i) := BEL(P_i) \cup \{\#(X)\}$

description: This action is used to verify that time-stamp X is fresh.

11. Abort

condition: Protocol run is illegal

result: Analysis reports failure

description: This could happen under various circumstances where there is an inconsistency or other flaw in the protocol specification.

12. Generate-key-pair(k^+, k^-)

result: $POSS(P_i) := POSS(P_i) \cup$
$\{k^+ \bowtie P_i, k^- \bowtie P_i\}$

description: This action is used to generate to keys that are the inverses of each other.

13. Apply-asymkey(X, k)

condition: $X, k \in POSS(P_i)$, k is an asymmetric key.

result: **if** $X = \{Y\}_{k'}$ **and** k is the inverse of k', **then** $POSS(P_i) := POSS(P_i) \cup$ Y **else** $POSS(P_i) := POSS(P_i) \cup \{\{X\}_k\}$

description: States that if an asymmetric key operation is performed on X, then there are two cases. Either X is of the form $\{Y\}_{k'}$ and k and k' are inverses, in which case the two keys cancel each other, or X is encrypted under k.

14. Bind(k, P_j)

condition: $k \in POSS(P_i)$, k is a key intended for P_j.

result: $POSS(P_i) := POSS(P_i) \cup \{(k \bowtie P_j)\}$, $Bindings(P_i) := Bindings(P_i) \cup$ $\{(k \bowtie P_j)\}$

description: States that if P_i possesses k, then after binding it to P_j, P_i possesses $k \bowtie P_j$. It is used to bind a key to a principal before sending it. $k \bowtie P_j$ is added to the *Bindings* set of P_i.

APPENDIX B

SETS

B.1 Global Sets

Principal Set: This set contains the principals who participate in a protocol. $P = \{P_1, P_2, \cdots, P_n\}$. Any P_i may be marked as an initiator of the protocol. We will assume there is only one initiator.

Rule Set: This set contains inference rules for deriving new statements from existing assertions. These are the same as the inference rules in the BAN logic. $R = \{R_1, R_2, \cdots, R_n\}$ where R_i is of the form $\frac{C_1, C_2, \cdots, C_n}{D}$, C_i is a condition and D is a statement.

Secret Set: This set contains all of the secrets that exist at any given time in the system. The cardinality of this set changes during the analysis as new secrets, such as session keys, are added. $S = \{S_1, S_2, \cdots, S_n\}$.

Observers Sets: For each S_i, $Observers(S_i)$ contains all the principals who could possibly know the secret S_i by listening to network traffic or generating it themselves. The members of the *Observers* sets can be stated explicitly or maintained as formulas representing their membership.

B.2 Local Sets

Possession Set(P_i) This set contains all the data relevant to security that this principal knows or possesses. This includes secret encryption keys, public keys, data that must remain secret, and any other information that is not publicly available. $POSS(P_i) = \{poss_1, poss_2, \cdots, poss_n\}$. $poss_i$ contains two fields: the actual data and the origin of the data.

Belief Set(P_i) This set contains all the beliefs held by a principal. This includes the belief that the keys it holds between itself and other principals are good, beliefs

about jurisdiction, beliefs about freshness, and beliefs about the possessions of other principals. $BEL(P_i) = \{bel_1, bel_2, \cdots, bel_n\}$.

Opaque(P_i) This set contains candidates to be added to the seen set. It is used by the `Update` function. The set contains plaintext message parts and a list of the associated keys needed to see them.

Seen(P_i) This set contains plaintext message parts that P_i sees from messages sent across the network. The seen sets collectively contain the same information as the observers sets.

Haskeys(P_i) This set contains keys that P_i sees either because they are in the initial possession set, or because they appear in a message sent across the network and are added to P_i's seen set.

Behavior List(P_i) This item is a list rather than a set because the elements are ordered. $BL = \{AL, bvr_1, bvr_2, \cdots, bvr_n\}$. AL is an action list.

Bindings Set(P_i) This set contains the legal bindings of keys held by a principal. These are bindings that are created by P_i, and bindings that are received in certificates from trusted servers. $Bindings(P_i) = \{k_1 \bowtie P_1, k_2 \bowtie P_2, \cdots, k_n \bowtie P_n\}$.

APPENDIX C

INFERENCE RULES

C.1 Nonce Verification Rule:

$$\frac{\#(X) \in BEL(P),\ X \text{ from } Q \in POSS(P)}{BEL(P) := BEL(P) \cup \{Q \text{ believes } \#(X)\}}$$

C.2 Message Meaning Rule:

$$\frac{\{X\}_k \text{ from } Q \in POSS(P),\ k \in POSS(P)}{BEL(P) := BEL(P) \cup \{X \in POSS(Q)\}}$$

C.3 Submessage Freshness Rule:

$$\frac{\begin{array}{c}\#(x_1) \in BEL(P), \\ \{X \text{ contains } x_1, X \text{ contains } x_2\} \subseteq POSS(P)\end{array}}{BEL(P) := BEL(P) \cup \#(x_2)}$$

C.4 Linkage Rule (symmetric keys):

$$\frac{\begin{array}{c}\#(k) \in BEL(P), k \in POSS(P), \\ LINK(N_a) \in BEL(P), X \text{ contains } f(N_a), \\ X \text{ contains } x_1, \{X\}_k \text{ from } Q \in POSS(P)\end{array}}{BEL(P) := (BEL(P) \Leftrightarrow LINK(N_a)) \cup \{\#(x_1)\}}$$

C.5 Linkage Rules (asymmetric keys):

$$\frac{\begin{array}{c}\#(k^-) \in BEL(P), k^- \in POSS(P), \\ LINK(N_a) \in BEL(P), X \text{ contains } f(N_a), \\ X \text{ contains } x_1, \{X\}_{k^+} \text{ from } Q \in POSS(P)\end{array}}{BEL(P) := (BEL(P) \Leftrightarrow LINK(N_a)) \cup \{\#(x_1)\}}$$

$$\frac{\#(k^+) \in BEL(P), k^+ \in POSS(P), \\ LINK(N_a) \in BEL(P), X \text{ contains } f(N_a), \\ X \text{ contains } x_1, \{X\}_k\text{- from } Q \in POSS(P)}{BEL(P) := (BEL(P) \Leftrightarrow LINK(N_a)) \cup \{\#(x_1)\}}$$

C.6 Possible Origins Rule:

$$\frac{X \in POSS(P), X \text{ contains } x_1, R \in Obs(x_1), R \neq P}{x_1 \text{ from } R \in POSS(P)}$$

C.7 Submessage Origin Rule:

$$\frac{\{X\}_{k^+} \in POSS(P), X \text{ contains } x_1 \text{ from } Q, \\ R \in Observers(k^-), X \text{ contains } x_2, R \neq P}{x_2 \text{ from } Q \in POSS(P), x_2 \text{ from } R \in POSS(P)}$$

C.8 Submessage Origin Rule for Public Keys:

$$\frac{\{X\}_{k_p{}^+} \in POSS(P), \\ X \text{ contains } x_1 \text{ from } Q, X \text{ contains } x_2}{x_2 \text{ from } Q \in POSS(P)}$$

C.9 Submessage Origin Rule for Private Keys:

$$\frac{\{X\}_{k_q}\text{-} \in POSS(P), X \text{ contains } x_2}{x_2 \text{ from } Q \in POSS(P)}$$

C.10 Certificate binding rule

$$\frac{(k \bowtie Q) \text{ from } S \in POSS(P), Trusted(S) \in BEL(P)}{Bindings(P) := Bindings(P) \cup \{(k \bowtie Q)\}}$$

C.11 Unbound key rule

$$\frac{k \in POSS(P), \nexists Q : (k \bowtie Q) \in POSS(P)}{Observers(k) := \mathcal{W}}$$

C.12 Bound key origin rule for symmetric keys

$$\frac{(k \bowtie Q) \in POSS(P), Q \neq P, \quad (k \bowtie Q) \notin Bindings(P)}{Abort}$$

C.13 Bound key origin rules for asymmetric keys

$$\frac{(k^{+} \bowtie Q) \in POSS(P), (k^{+} \bowtie Q) \notin Bindings(P), \quad \{X\}_{k^{-}} from R \in POSS(P), R \neq Q}{Abort}$$

$$\frac{(k^{-} \bowtie Q) \in POSS(P), (k^{-} \bowtie Q) \notin Bindings(P), \quad \{X\}_{k^{+}} from R \in POSS(P), R \neq Q}{Abort}$$

APPENDIX D

THE CORRECTED KHAT PROTOCOL

Client

$POSS(C) = \{K\}$

$BEL(C) = \{\#(K)\}$

$BL(C) =$

Part of ticket granting

Receive(S,$\{N_1, N_2, \cdots N_n\}$)

Split($\{N_1, N_2, \cdots N_n\}$)

Phase I

Generate-secret(SF)

Generate-secret(N)

Encrypt(K,N)

Send(S,Encrypt(Concat(SF, N, N_i), K))

Update($\{SF \cdot N \cdot N_i\}_K$)

Forget-secret(N)

Forget-secret(SF)

Phase II

Receive($S, \{N \cdot \{SF \cdot TGT_C\}_K\}$)

Split($\{N \cdot \{SF\}_K\}$)

Decrypt($\{K\}_N, N$)

Decrypt($\{SF \cdot TGT_C\}_K, K$)

Split($\{SF \cdot TGT_C\}$)

Check-freshness(TGT_C)

Server

$POSS(S) = \{K\}$

$BEL(S) = \{\#(K)\}$

$BL(S) =$

Part of ticket granting

- Generate-nonce(N_1)

  Generate-nonce(N_2)

  $\cdots$

  Generate-nonce(N_n)

  Send(C,Concat($N_1, N_2, \cdots, N_n$))

  Update($\{N_1, N_2, \cdots N_n\}$)

Phase I

Receive($C, \{SF \cdot N \cdot N_i\}_K$)

Decrypt($\{SF \cdot N \cdot N_i\}_K, K$)

Split($\{SF \cdot N \cdot N_i\}$)

Phase II

Generate-secret(TGT_C)

Send(Concat(N,Encrypt(Concat(SF, TGT_C), K)))

Update($\{N \cdot \{SF \cdot TGT_C\}_K\}$)